Tabla de contenidos

1. Entornos Operativos

1.1. Objetivo

En esta lección el alumno conocerá las herramientas de mayor difusión en el mercado laboral, al finalizar la misma estará en condiciones de poder conectarse desde cualquiera de las herramientas, visualizar sus paneles de interacción.

A lo largo del curso se utilizaran en forma alternativa las distintas herramientas que mencionamos.

1.1.1. Herramientas Provistas por Oracle

Hasta el momento ORACLE provee tres herramientas de uso libre, dos vienen con la instalación del servidor o cliente y la tercera se baja directamente del sitio de ORACLE.

- •SQL*Plus
- •iSQL*Plus
- •SQL Developer
- SQL*Plus

Es una herramienta de entorno carácter, viene desde la primera versión de ORACLE y se mantiene hasta el presente.

Si bien es una herramienta que pareciera obsoleta, ante distintas dificultades como falta de licencias para otros programas, imposibilidad de instalar, o cualquier dificultad que se nos presente, siempre podremos trabajar con esta herramienta.

1.1.2. iSQL*Plus

Esta herramienta fue incorporada a partir de la versión ORACLE 9i es una herramienta disponible en web con características visuales superiores a su antecesor SQL*Plus. Esta herramienta no es muy utilizada por las empresas, dado que a nivel operativo utilizan normalmente TOAD o Navigator.

1.1.3. SQL Developer

Es una herramienta de libre distribución que se puede bajar desde http://www.oracle.com/technology/software/products/sql/index.html para ello hay que registrarse en forma gratuita y se baja directamente.

Esta herramienta esta desarrollada en Java, utilizando el conector de jdbc para conectarse a la base, motivo por el cual no necesita cliente ORACLE instalado.

El estar hecho en java mejoro notablemente la calidad gráfica de la herramienta e hizo una herramienta muy intuitiva.

1.2. Herramientas provistas por terceros

Existen en el mercado diversas aplicaciones desarrolladas por terceros con uso bajo licencia, de ellas se destacan:

- TOAD
- SQL Navigator
- PL/SQL Developer

Tanto TOAD como SQL Navigator son desarrolladas por Quest Software, ambas son de entorno gráfico muy intuitivas al utilizarse con funciones para desarrolladores como para DBA

1.2.1. PL/SQL Developer

Es desarrollada por Arround Automations en un principio su finalidad fue destinada al trabajo de unidades procedimentales, en mi experiencia contaba con un debug excelente, cosa que superaba a TOAD.

También es una herramienta de uso bajo licencia.

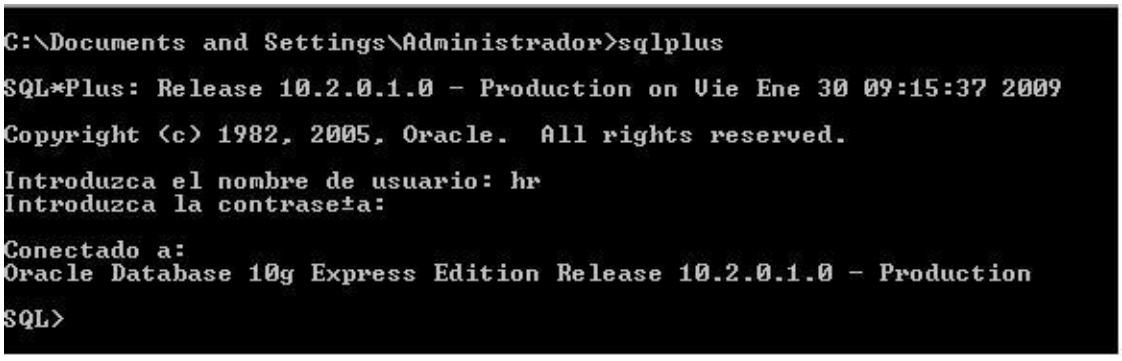
1.2.2. SQL*Plus

Si bien como dijimos es una herramienta de entorno carácter muy poco sensitiva, todavía hay un hecho que indique que debemos tenerla siempre en cuenta. Esto es que se conecta directamente al servidor, sin la necesidad de tomar configuraciones regionales que pueden confundir el formato de los datos, como ser formato de fechas, puntos en lugar de comas para la separación decimal, etc. Por otro lado para el DBA sigue siendo fundamental a la hora de tener que realizar trabajos sobre la base de datos dado a que con SQL*Plus puede parar o arrancar la base, cosa que no puede con las de entorno gráfico.

1.3. Conexión de usuarios

Desde el prompt tanto de Windows como Linux se hace la llamada al programa

C:>sqlplus



Al ingresar pide el nombre de usuario y la password correspondiente una vez ingresado si es valido me indica que estoy conectado e identifica sobre que tipo, versión y release voy a trabajar.

Otra forma de conectarse es ingresando desde el prompt la cadena de conexión de la siguiente forma:

C:>sqlplus usuario@base_de_datos

Una vez ingresado los valores nos pide la password y repite lo mostrado en la imagen anterior

1.3.1. Configuraciones Básicas

Existen dos configuraciones que son esenciales para trabajar cómodos.

La primera es ajustar la ventana de Windows desde la pestaña de propiedades para garantizar el ancho y alto adecuados.

La segunda hace al entorno de SQL*Plus donde hay que ejecutar un comando:

SET LINESIZE 32000

Este comando configura la cantidad de caracteres máxima que va a mostrar en una sola línea, si el tamaño de seteo de LINESIZE es menor la salida que mostrará lo hará en dos líneas dificultando la identificación de los datos.

1.4. SQL Developer

Esta herramienta no requiere instalación una vez descargada de la página, se descomprime en su ubicación y finaliza el proceso de instalación.

Para ejecutarla hay que buscar archivo sqldeveloper.exe hacer click y el programa es lanzado.

1.4.1. Configuración

Para conectarse a una base de datos primero hay que realizar la configuración correspondiente del conector jdbc, para ello debemos obtener una serie de datos que tendremos que solicitar al DBA.

Hacer click en nueva conexión y se abrirá una ventana donde nos pide una serie de valores:

Conection Name	Es el nombre que nosotros le daremos a la conexión
User Name	Nombre de usuario
Password	Clave con la que fue creado el usuario
Role	Para un usuario común el role es 'Default'
Hostname	Nombre o IP del servidor donde se encuentra la base de datos
Port	Port donde se conecta la base de datos
SID	Nombre de la base de datos

Para confirmar la validez de los datos ingresados oprimir el botón TEST si existió algún error lo mostrara en color rojo indicando el error.

1.4.2. TOAD

Esta herramienta necesita que haya instalado un cliente Oracle en la máquina donde va a ejecutarse.

1.5. Conexión a base de datos

Necesita el cliente Oracle instalado y levanta los valores del archivo tnsnames que se encuentra en el path ORACLE_HOME\NETWORK\ADMIN

Es un archivo de texto editable en el ejemplo se muestra una configuración de tnsname.

```
XE =
  ( DESCR I PT ION =
                ADDRESS = (PROTOCOL = TCP ) (HOST = desk top ) (PORT
= 1521 ) )
                                        (CONNECT_DATA =
                                                (SERVER = DED ICATED )
                                                ( SERV ICE_NAME = XE )
                                        )
        )
```

XE es el nombre con que identifique la conexión no necesariamente se debe llamar como la base de datos.

HOST: indica el nombre o IP donde esta corriendo la base de datos

PORT: Donde se conecta a la base de datos

SERVICE_NAME: El nombre de la base de datos.

Con estos valores TOAD se puede conectar a la base de datos

1.5.1. Crear una conexión de usuario

Desde la opción SESSION del menú se selecciona NEW SESSION, se abre una nueva ventana donde se ingresarán los valores de:

Usuario y Password una lista desplegable nos permite seleccionar la base a la que vamos a conectarnos, si tildamos en la parte inferior el CHECK SAVE PASSWORD queda guardada la conexión.

A partir de allí podremos comenzar a trabajar con la base de datos.

2. Sentencias SQL Básicas

2.1. Objetivos

Para extraer datos de la base de datos, debe utilizar la sentencia SELECT de SQL (Lenguaje Estructurado de Consulta). Puede que tenga que restringir las columnas que se van a mostrar. Esta lección describe todas las sentencias SQL necesarias para realizar estas acciones. Es posible que desee crear sentencias SELECT que se puedan utilizar más de una vez.

2.1.1. Capacidades de las Sentencias SELECT de SQL

Las sentencias SELECT recuperan información de la base de datos.

Con las sentencias SELECT puede utilizar estas capacidades:

- Proyección: Seleccione las columnas de una tabla que se han devuelto mediante una consulta. Elija la cantidad de columnas que necesite.
- Selección: Seleccione las filas de una tabla que se han devuelto mediante una consulta. Se pueden utilizar varios criterios para restringir las filas que se recuperarán.
- Unión: Junte datos almacenados en diferentes tablas especificando el enlace que hay entre ellos.

2.1.2. Sentencia SELECT Básica

En su forma más sencilla, una sentencia SELECT debe incluir:

- Una cláusula SELECT, que especifica las columnas que se van a mostrar
- Una cláusula FROM, que identifica la tabla que contiene las columnas que se muestran en la cláusula SELECT

En la sintaxis:

Palabras claves y Cláusulas	Definición
SELECT	Es una lista de una o más columnas
*	Selecciona todas las columnas
DISTINCT	No visualiza registros duplicados
column|expression	Selecciona la columna o la expresión especificadas
Alias	Proporciona a las columnas seleccionadas cabeceras diferentes
FROM table	Especifica la tabla que contiene las columnas

Nota: A lo largo de este curso, las expresiones palabra clave, cláusula y sentencia se utilizan como se indica a continuación:

- Una palabra clave hace referencia a un elemento SQL individual.

Por ejemplo, SELECT y FROM son palabras clave.

- Una cláusula es una parte de una sentencia SQL.

Por ejemplo, SELECT employee_id, last_name, ... es una cláusula.

- Una sentencia es una combinación de dos o más cláusulas.

Por ejemplo, SELECT * FROM employees es una sentencia SQL

2.1.3. Selección de Todas las Columnas de Todas las Filas

Puede mostrar todas las columnas de datos de una tabla si agrega un asterisco (*) después de la palabra clave SELECT. En el ejemplo de la diapositiva, la tabla DEPARTMENTS contiene cuatro columnas: DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID y LOCATION_ID. La tabla contiene siete filas, una para cada departamento.

También puede mostrar todas las columnas de la tabla enumerando todas las columnas después de la palabra clave SELECT. Por ejemplo, la siguiente sentencia SQL muestra todas las columnas y todas las filas de la tabla DEPARTMENTS:

```
SELECT    department_id, department_name,  manager_id,
            location_id
FROM      departments;
```

2.1.4. Selección de Columnas Específicas de Todas las Filas

Puede utilizar la sentencia SELECT para mostrar columnas específicas de la tabla especificando los nombres de columna, separados por comas. El ejemplo de la diapositiva muestra todos los números de departamento y los números de ubicación de la tabla DEPARTMENTS.

En la cláusula SELECT, especifique las columnas que desee, en el orden en que quiere que aparezcan en la salida. Por ejemplo, para que la ubicación se muestre antes del número de departamento de izquierda a derecha, utilice esta sentencia:

```
SELECT location_id, department_id
FROM      departments;
```

2.1.5. Escritura de Sentencias SQL

Mediante las sencillas reglas que se detallan a continuación, puede crear sentencias válidas que resulten fáciles de leer y de editar:

- Las sentencias SQL no son sensibles a mayúsculas/minúsculas. (a menos que se indique que lo sean).
- Las sentencias SQL se pueden introducir en una o en varias líneas.
- Las palabras clave no se pueden dividir en líneas ni se pueden abreviar.
- Las cláusulas se suelen colocar en líneas aparte para facilitar su lectura y su edición.
- Se deben utilizar sangrados para facilitar la lectura del código.

- Las palabras clave se suelen introducir en mayúsculas; el resto de palabras, como nombres y columnas, se introduce en minúsculas.

2.1.6. Valores por Defecto de Cabeceras de Columnas

SQL*Plus - TOAD - SQL Developer:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División

Las cabeceras de las columnas CHARACTER y DATE están alineadas a la izquierda

Las cabeceras de columnas NUMBER están alineadas a la derecha

Visualización de cabecera por defecto: mayúsculas

2.2. Expresiones Aritméticas

Es posible que deba modificar el modo en que se muestran los datos o que desee realizar cálculos o consultar supuestos hipotéticos. Todo esto es posible mediante expresiones aritméticas.

Una expresión aritmética puede contener nombres de columna, valores numéricos constantes y los operadores aritméticos.

2.2.1. Operadores Aritméticos

Puede utilizar operadores aritméticos en cualquier cláusula de una sentencia SQL (excepto en la cláusula FROM).

Nota: Con los tipos de datos DATE, puede utilizar únicamente los operadores de suma y de resta.

Se puede utilizar el operador de suma para calcular una aumento en el salario de 300 pesos para cada empleado.

Nota: Oracle Server ignora los espacios en blanco que van antes y después del operador aritmético.

```
SELECT last_name, salary, salary + 300
FROM   employees;
```

2.2.2. Prioridad de Operadores

Si un operador aritmético contiene más de un operador, la multiplicación y la división se evalúan en primer lugar. Si los operadores de una expresión tienen la misma prioridad, la evaluación se realiza de izquierda a derecha.

Puede utilizar paréntesis para forzar a que la expresión que vaya entre paréntesis se evalúe primero.

2.2.3. Reglas de Prioridad:

- La multiplicación y la división tienen lugar antes que la suma y la resta.
- Los operadores con la misma prioridad se calculan de izquierda a derecha.
- Los paréntesis se utilizan para omitir la prioridad por defecto o para aclarar la sentencia

Nota: Utilice paréntesis para reforzar el orden estándar de prioridad y para mejorar la claridad. Por ejemplo, la expresión de la diapositiva se puede escribir como (12*salary)+100 sin que cambie el resultado

2.2.4. Valores Nulos

Si faltan valores en una fila para una columna en particular, se dice que el valor es nulo o que contiene un valor nulo.

Un valor nulo es aquel que no está disponible, no está asignado, es desconocido o no es

aplicable. Un valor nulo no es lo mismo que un cero o un espacio. El cero es un número y un espacio es un carácter.

Las columnas de cualquier tipo de datos pueden contener valores nulos. Sin embargo, algunas restricciones (NOT NULL y PRIMARY KEY) impiden que se utilicen valores nulos en la columna.

2.2.5. Valores Nulos en Expresiones Aritméticas

Si el valor de alguna columna de una expresión aritmética es nulo, el resultado es nulo. Por ejemplo, si intenta realizar una división por cero, recibirá un error. Sin embargo, si divide un número por un valor nulo, el resultado es un valor nulo o desconocido.

2.2.6. Alias de Columna

Al mostrar el resultado de una consulta, se utiliza normalmente el nombre de la columna seleccionada como cabecera de columna. Esta cabecera puede no ser descriptiva y, por tanto, puede resultar difícil de entender. Puede cambiar una cabecera de columna

mediante un alias.

Especifique el alias después de la columna en la lista SELECT utilizando un espacio como

separador. Por defecto, las cabeceras de alias aparecen en mayúsculas. Si el alias contiene espacios o caracteres especiales (como # o $) o si es sensible a mayúsculas/minúsculas, ponga el alias entre comillas dobles (" ").

- Cambia el nombre de una cabecera de columna
- Es útil para los cálculos
- Sigue inmediatamente al nombre de columna (puede haber también una palabra clave AS
- opcional entre el nombre de columna y el alias)
- Requiere comillas dobles si contiene espacios o caracteres especiales, o si es sensible a mayúsculas/minúsculas

2.2.7. Operador de Concatenación

Puede enlazar columnas a otras columnas, expresiones aritméticas o valores constantes para crear una expresión de carácter mediante el operador de concatenación (||). Las columnas situadas en cualquiera de los lados del operador se combinan para crear una única columna de salida.

En el ejemplo, LAST_NAME y JOB_ID están concatenadas y se les ha asignado el alias

Employees. Observe que el apellido y el código de puesto del empleado están combinados para crear una única columna de salida.

La palabra clave AS antes del nombre de alias facilita la lectura de la cláusula SELECT. Valores Nulos con el Operador de Concatenación

Si se concatena un valor nulo con una cadena de caracteres, el resultado es una cadena de caracteres. LAST_NAME || NULL da como resultado LAST_NAME.

2.2.8. Cadenas de Literales

Un literal es un carácter, un número o una fecha que se ha incluido en la lista SELECT y que no es un nombre de columna ni un alias de columna. Se imprime con cada fila devuelta. Las cadenas de literales de texto sin formato se pueden incluir en el resultado de la consulta y se tratarán igual que una columna en la lista SELECT.

Los literales de fecha y de caracteres deben ir entre comillas simples (' '); los literales de números no es necesario que vayan entre comillas.

2.2.9. Operador de Comillas (q) Alternativo

Muchas sentencias SQL utilizan literales de caracteres en expresiones o condiciones. Si el literal contiene una comilla simple, puede utilizar el operador de comillas (q) y seleccionar su propio delimitador de comillas.

Puede seleccionar cualquier delimitador apropiado, de un solo byte o multibyte, o cualquiera de los siguientes pares de caracteres: [ ], { }, ( ) o < >.

En el ejemplo que se muestra, la cadena contiene comillas simples, lo que normalmente se interpreta como delimitador de una cadena de caracteres. Pero mediante el operador q, los corchetes [] se utilizan como delimitador de comillas. La cadena situada entre los

delimitadores de corchetes se interpreta como cadena de literales.

```
SELECT depa r t men t_name | | q ' [ , i t 's ass igned Manage r Id : ] '
| |
                manager_id     AS "Depar tment and Manage r"
FROM depa r tment s ;
```

2.2.10. Filas Duplicadas

A menos que indique lo contrario, se muestra los resultados de una consulta sin eliminar las filas duplicadas.

Para eliminar filas duplicadas del resultado(No borra registros), incluya la palabra clave DISTINCT de la cláusula SELECT justo después de la palabra clave SELECT.

Puede especificar varias columnas después del cualificador DISTINCT.

El cualificador DISTINCT afecta a todas las columnas seleccionadas y el resultado son todas las combinaciones distintas de las columnas.

```
SELECT      DISTINCT department_id, job_id
FROM        employees;
```

2.2.11. Limitación de las Filas que se Seleccionan

Puede restringir las filas que se devuelven desde la consulta mediante la cláusula WHERE. Una cláusula WHERE contiene una condición que se debe cumplir y sigue directamente a la cláusula FROM. Si la condición es verdadera, se devuelve la fila que cumple la condición.

```
SELECT * | { [ DISTINCT ] column | expression [alias ] , . . . }
FROM          table
[ WHERE condition(s ) ] ;
```

En la sintaxis:

WHERE	Restringe la consulta a las filas que cumplan una condición
condition	Se compone de nombres de columna, expresiones, constantes y un operador de comparación

La cláusula WHERE puede comparar valores en columnas, valores de literales, expresiones aritméticas o funciones. Consta de tres elementos:

- Nombre de columna
- Condición de comparación
- Nombre de columna, constante o lista de valores

2.2.12. Uso de la Cláusula WHERE

En el ejemplo, la sentencia SELECT recupera el identificador de empleado, el nombre, el

identificador de puesto y el número de departamento de todos los empleados que están en el departamento 90.

```
SELECT emp l oyee_ id , l as t_name , job_ id , depa r t men t_ id FROM
emp loyees
WHERE      depa r tmen t_ id = 90 ;
```

2.2.13. Cadenas de Caracteres y Fechas

Las cadenas de caracteres de la cláusula WHERE deben ir entre comillas simples ('').

Sin embargo, las constantes numéricas no deben ir entre comillas simples.

Todas las búsquedas de caracteres son sensibles a mayúsculas/minúsculas. En el ejemplo siguiente, no se devuelve ninguna fila porque la tabla EMPLOYEES almacena los apellidos en caracteres de mayúsculas/minúsculas mezclados.

```
SELECT  las t_name,   j ob_ id , depa r t men t_ id
FROM          emp loyees
WHERE      l as t_name = ' WHALEN ' ;
```

2.3. Condiciones de Comparación

Las condiciones de comparación se utilizan en condiciones que comparan una expresión con otro valor u otra expresión. Se utilizan en la cláusula WHERE en el siguiente formato:

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<> != y ^=	Distinto de
BETWEEN...AND...	Entre dos valores (ambos inclusive)
IN(set)	Se corresponde con cualquier valor de una lista
LIKE	Se corresponde con un patrón de caracteres
IS NULL	Es un valor nulo

Sintaxis

.. . WHERE exp r ope ra to r va lue

E jemp lo

.. . WHERE h i r e_da te = ' 01 - JAN -95 ' .. . WHERE sa la ry >= 6000

.. . WHERE l as t_name = ' Sm i th '

No se puede utilizar un alias en la cláusula WHERE.

Nota: Los símbolos != y ^= también pueden representar la condición no igual que.

2.3.1. Uso de Condiciones de Comparación

En el ejemplo, la sentencia SELECT recupera el apellido y el salario de la tabla EMPLOYEES de cualquier empleado cuyo salario es menor o igual que 3.000 pesos.

Observe que hay un valor explícito suministrado para la cláusula WHERE. El valor explícito 3000 se compara con el valor de salario de la columna SALARY de la tabla EMPLOYEES.

```
SELECT last_name, salary
FROM   employees
WHERE  salary<= 3000;
```

2.3.2. Uso de la Condición BETWEEN

Puede mostrar filas basándose en un rango de valores mediante la condición de rango BETWEEN. El rango que especifique contiene un límite inferior y uno superior.

La sentencia SELECT del ejemplo devuelve filas de la tabla EMPLOYEES de cualquier empleado cuyo salario esté entre los 2.500 y los 3.500 dólares.

Los valores que se especifican con la condición BETWEEN también se incluyen. Debe especificar el límite inferior en primer lugar.

También puede utilizar la condición BETWEEN en valores de caracteres:

```
SELECT last_name
FROM   employees
WHERE  last_name BETWEEN 'King' AND 'Smith';
```

2.3.3. Uso de la Condición IN

Para probar valores en un juego de valores especificado, utilice la condición IN. La condición IN se conoce también como condición de miembro.

El ejemplo de la diapositiva muestra los números de empleado, los apellidos, los salarios y los números de empleado de supervisor para todos los empleados cuyo número de empleado de supervisor sea 100, 101 ó 201.

La condición IN se puede utilizar con cualquier tipo de datos. El ejemplo siguiente devuelve una fila de la tabla EMPLOYEES para cualquier empleado cuyo apellido se incluya en la lista de nombres de la cláusula WHERE:

```
SELECT employee_id, manager_id, department_id
```

```
FROM        employees
WHERE    last_name IN ('Hartstein', 'Vargas');
```

Si los caracteres o las fechas se utilizan en la lista, deben ir entre comillas simples ('').

2.3.4. Uso de la Condición LIKE

Puede que no siempre sepa qué valor buscar exactamente. La condición LIKE permite seleccionar filas que se correspondan con un patrón de caracteres. La operación de

correspondencia de patrones de caracteres se conoce como búsqueda con comodines.

Se pueden utilizar dos símbolos para crear la cadena de búsqueda.

Símbolo	Descripción
%	Representa cualquier secuencia de cero o más caracteres
_	Representa cualquier carácter simple

La sentencia SELECT del ejemplo devuelve el nombre de empleado de la tabla EMPLOYEES de cualquier empleado cuyo nombre empiece por S. Observe la S en mayúsculas. Los nombres que empiecen por s no se devolverán.

La condición LIKE se puede utilizar como método abreviado para algunas comparaciones BETWEEN. El ejemplo siguiente muestra los apellidos y las fechas de contratación de todos los empleados que se hayan incorporado entre enero de 1995 y diciembre de 1995:

```
SELECT last_name, hire_date
FROM        employees
WHERE    hire_date LIKE '%95';
```

Cuando necesite una correspondencia exacta de los propios caracteres % y _, utilice la opción ESCAPE. Esta opción especifica qué es el carácter de escape. Si desea buscar las cadenas que contienen 'SA_', puede utilizar esta sentencia SQL:

```
SELECT employee_id, last_name, job_id
FROM        employees
WHERE    job_id LIKE '%SA\_%' ESCAPE '\';
```

2.3.5. Uso de las Condiciones NULL

Las condiciones NULL son la condición IS NULL y la condición IS NOT NULL.

La condición IS NULL comprueba si hay valores nulos. Un valor nulo significa que el valor no está disponible, no está asignado, es desconocido o no es aplicable. Por tanto, no puede probar con = porque un valor nulo no puede ser igual ni desigual a ningún valor.

El ejemplo recupera los apellidos y los supervisores de todos los empleados que no

tienen supervisor.

```
SELECT last_name, job_id, commission_pct
FROM   employees
WHERE  commission_pct IS NULL;
```

2.3.6. Condiciones Lógicas

Una condición lógica combina el resultado de dos condiciones componentes para crear un único resultado basándose en esas condiciones, o invierte el resultado de una sola condición. Se devuelve una fila sólo si el resultado global de la condición esverdadero.

En SQL, hay disponibles tres operadores lógicos:

- AND
- OR
- NOT

Hasta ahora, todos los ejemplos han especificado únicamente una condición en la cláusula WHERE. Puede utilizar varias condiciones en una cláusula WHERE mediante los operadores AND y OR.

2.4. Uso del Operador AND

En el ejemplo, ambas condiciones deben ser verdaderas para que se seleccione algún registro. Por tanto, sólo se seleccionan los empleados que tengan un cargo que contenga la cadena 'MAN' y que ganen 10.000 pesos o más.

Todas las búsquedas de caracteres son sensibles a mayúsculas/minúsculas. No se devuelve ninguna fila si 'MAN' no está en mayúsculas. Las cadenas de caracteres deben ir entre comillas.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE salary >=10000
AND job_ id LIKE '%MAN%';
```

2.4.1. Uso del Operador OR

En el ejemplo, una de las condiciones debe ser verdadera para que se seleccione algún

registro. Por tanto, se selecciona cualquier empleado que tenga un identificador de puesto que contenga la cadena 'MAN' o que gane 10.000 dólares o más.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary >= 10000
OR job_ id LIKE '%MAN%';
```

2.4.2. Uso del Operador NOT

El ejemplo se muestra el apellido y el identificador de puesto de todos los empleados cuyo identificador de puesto no es IT_PROG, ST_CLERK o SA_REP.

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP');
```

El operador NOT también se puede utilizar con otros operadores SQL, como BETWEEN, LIKE y NULL.

```
. . . WHERE      job_id      NOT   IN ('AC_ACCOUNT', 'AD_VP')
. . . WHERE      salary      NOT      BETWEEN        10000 AND      15000
. . . WHERE      last_name   NOT      LIKE '%A%'
... WHERE      commission_pct      IS          NOT      NULL
```

2.5. Reglas de Prioridad

Las reglas de prioridad determinan el orden en que se evalúan y se calculan las expresiones. La tabla muestra el orden de prioridad por defecto. Puede sustituir este orden por defecto poniendo entre paréntesis las expresiones que desee calcular primero.

Operador	Significado
1	Operadores aritméticos
2	Operador de concatenación
3	Condiciones de comparación
4	IS [NOT] NULL, LIKE,[NOT] IN
5	[NOT] BETWEEN
6	Distinto de
7	Condición lógica NOT
8	Condición lógica AND
9	Condición lógica OR

2.5.1. Uso de la Cláusula ORDER BY

El orden de las filas que se devuelven en el resultado de una consulta no está definido. Se puede utilizar la cláusula ORDER BY para ordenar las filas. Si utiliza la cláusula ORDER BY, debe ser la última cláusula de la sentencia SQL.

Puede especificar una expresión, un alias o una posición de columna como condición de ordenación.

Sintaxis

```
SELECT expr
```

```
FROM tabl e
[ WHERE condi t ion (s ) ]
[ORDER BY {co lumn, expr, numer ic_posi t ion} [ASC|DESC][NULLS
LAST  NULLS F IRST ] ] ;
```

En la sintaxis:

- ORDER BY especifica el orden en el que se muestran las filas recuperadas
- ASC ordena las filas en orden ascendente (ordenación por defecto)
- DESC ordena las filas en orden descendente
- NULLS LAST – NULLS FIRST el orden en que se mostrarán los valores nulos

Si no se utiliza la cláusula ORDER BY, el orden no se define y Oracle Server puede norecuperar las filas en el mismo orden para la misma consulta dos veces. Utilice la cláusula

- ORDER BY para mostrar las filas en un orden específico.

2.5.2. Ordenación de Datos por Defecto

El orden por defecto es ascendente:

Los valores por defecto se muestran con los valores más bajos en primer lugar (por ejemplo, de 1 a 999).

Los valores se muestran con el valor más antiguo en primero lugar (por ejemplo, 01-ENE-92 antes que 01-ENE-95).

Los valores de caracteres se muestran por orden alfabético (por ejemplo, A al principio y Z al final).

Los valores nulos se muestran al final en las secuencias ascendentes y al principio en las descendentes.

Puede ordenar por una columna que no esté en la lista SELECT.

2.5.3. Variables de Sustitución

Los ejemplos hasta ahora han sido codificados. En una aplicación finalizada, el usuario dispararía el informe y el informe se ejecutaría sin pedir al usuario que realice ninguna acción. El rango de datos lo predeterminaría le cláusula fija WHERE del archivo de comandos.

Para reutilizar el la cláusula SQL puede crear informes que pidan a los usuarios que suministren sus propios valores para restringir el rango de datos devueltos por variables de sustitución. Puede embeber variables de sustitución en un archivo de comandos o en una única sentencia SQL. Una variable se puede considerar un contenedor en el que los valores se almacenan temporalmente. Al ejecutarse la sentencia, el valor se sustituye.

Puede utilizar las variables de sustitución ampersand simple (&) para almacenar valores temporalmente.

Puede predefinir variables mediante el comando DEFINE . DEFINE crea y asigna un valor a una variable.

Ejemplos de Rangos Restringidos de Datos

- Información de cifras sólo para el trimestre actual o el rango de fechas especificado
- Información sobre datos relevantes únicamente para el usuario que solicita el informe
- Visualización de personal únicamente de un departamento dado

Otros Efectos Interactivos

Los efectos interactivos no se restringen a la interacción directa del usuario con la cláusula WHERE. Se pueden utilizar los mismos principios para alcanzar otros objetivos como,
por ejemplo:

- Obtener valores de entrada de un archivo y no de una persona
- Transferir valores de una sentencia SQL a otra

2.5.4. Variable de Sustitución Ampersand Simple &

Al ejecutar un informe, los usuarios a menudo desean restringir los datos que se devuelven dinámicamente. Utilice un ampersand (&) para identificar cada variable de la sentencia SQL. No es necesario que defina el valor de cada variable.

Notación	Descripción
&user_variable	Indica una variable en una sentencia SQL; si la variable no existe, iSQL*Plus pide al usuario un valor

Con el ampersand simple, se hace la petición al usuario cada vez que se ejecuta el comando, si no existe la variable.

```
SELECT employee_id, last_name, salary, department_id FROM      employees

WHERE    employee_id = &employee_num    ;
```

2.5.5. Especificación de Valores de Carácter y de Fecha con variables de Sustitución

En una cláusula WHERE, los valores de fecha y de carácter deben ir entre comillas simples.

Se aplica la misma regla a las variables de sustitución.

Ponga la variable entre comillas simples dentro de la propia sentencia SQL.

```
SELECT last_name, department_id, salary*12
FROM        employees
WHERE     job_id = '&job_title'       ;
```

2.5.6. Especificación de Nombres de Columna, Expresiones y Texto

No sólo puede utilizar las variables de sustitución en la cláusula WHERE de una sentencia SQL, sino que también las puede utilizar para sustituir nombres de columna, expresiones o texto.

Ejemplo

El ejemplo se muestra el número de empleado, el apellido, el cargo y cualquier otra columna que especifique el usuario en tiempo de ejecución, de la tabla EMPLOYEES.

Si no introduce un valor de sustitución, obtiene un error al ejecutar la sentencia anterior.

Nota: Una variable de sustitución se puede utilizar en cualquier parte de la sentencia SELECT, excepto como primera palabra en el prompt de comandos.

```
SELECT employee_id, last_name, job_id,&column_name
FROM   employees
WHERE  &condition
ORDER BY &order_column;
```

2.5.7. Variable de Sustitución Ampersand Doble &&

Puede utilizar la variable de sustitución ampersand doble (&&) si desea volver a utilizar el valor de variable sin pedir al usuario que realice una acción cada vez. El usuario verá el prompt para el valor sólo una vez. En el ejemplo, se pide al usuario que proporcione el valor para la variable column_name sólo una vez. El valor que suministra el usuario (department_id) se utiliza tanto para visualización como para la ordenación de los datos.

Se almacena el valor que se suministra mediante el comando DEFINE; lo vuelve a utilizar siempre que se haga referencia al nombre de variable. Cuando una variable de usuario está en su lugar, debe utilizar el comando UNDEFINE para suprimirlo de esta forma:

```
UNDEFINE column_name
SELECT   employee_id, last_name, job_id, &&column_name
FROM employees
ORDER BY &column_name;
```

2.6. Uso del Comando DEFINE de iSQL*Plus

El ejemplo que se muestra crea una variable de sustitución para un número de empleado mediante el comando DEFINE. En tiempo de ejecución, esto muestra el número de empleado, el apellido, el salario y el número de departamento de ese empleado.

Como la variable se crea mediante un comando DEFINE, no se pide al usuario que introduzca un valor para el número de empleado. En vez de eso, se sustituye automáticamente el valor de variable definido en la sentencia SELECT.

La variable de sustitución EMPLOYEE_NUM está presente en la sesión hasta que el usuario anule la definición o salga de la sesión .

```
DEF INE emp l oyee_num =200
SELECT emp l oyee_ id , l as t_name , sa la ry , depa r t men t_ id FROM
emp loyees
WHERE       emp loyee_ id = &emp loyee_num ;
UNDEF I NE emp loyee_num
```

2.6.1. Uso del Comando VERIFY

Para confirmar los cambios en la sentencia SQL, utilice el comando VERIFY. Al definir SET VERIFY ON se fuerza a que se muestre el texto de un comando antes y después de reemplazar con valores las variables de sustitución.

Variables de Sistema de SQL*Plus

SQL*Plus utiliza diversas variables de sistema que controlan el entorno de trabajo. Una de esas variables es VERIFY. Para obtener una lista completa de todas las variables de sistema, puede emitir el comando SHOW ALL.

2.7. Ejercicios

1. Cree un informe que muestre el apellido y el cargo de todos los empleados que no

tengan supervisor

2 El departamento de recursos humanos quiere ejecutar informes basados en un supervisor. Cree una consulta que pida al usuario un identificador de supervisor y genere el identificador de empleado, el apellido, el salario y el departamento de los

empleados de ese supervisor. El departamento de recursos humanos quiere poder ordenar el informe por una columna seleccionada. Puede probar los datos con estos valores:

3 Muestre el apellido de todos los empleados que tengan tanto una a como una e en su apellido

4. Muestre el apellido, el puesto de trabajo y el salario de todos los empleados que sean representante de ventas o administrativo y cuyo salario sea distinto de 2.500, 3.500 ó 7.000 pesos

5 Cree un informe para mostrar el apellido, el identificador de puesto y la fecha de inicio para los empleados con los apellidos Matos y Taylor. Ordene la consulta por orden ascendente por fecha de inicio.

3. Funciones de fila y Funciones generales Oracle

3.1. Objetivos

Las funciones hacen que el bloque de consulta sea más potente y se utilizan para manipular valores de datos. Ésta es la primera de dos lecciones que examinan las funciones. Se centra en las funciones de una sola fila de carácter, numéricas y de fecha, así como en las que convierten datos de tipo a otro (por ejemplo, conversión de datos de carácter a datos numéricos).

3.1.1. Funciones ORACLE

Las funciones son una característica muy potente de SQL. Se pueden utilizar para:

- Realizar cálculos en datos
- Modificar elementos de datos individuales
- Manipular la salida para grupos de filas
- Formatear fechas y números para su visualización
- Convertir tipos de datos de columnas

Las funciones SQL a veces toman argumentos y siempre devuelven un valor.

Nota: En su mayor parte, las funciones que se describen en esta lección son específicas de la versión Oracle de SQL.

Hay dos tipos de funciones:

- Funciones de una sola fila
- Funciones de varias filas

3.1.2. Funciones de una Sola Fila

Estas funciones operan sólo en filas únicas y devuelven un resultado por fila. Hay diferentes tipos de funciones de una sola fila. Esta lección trata las siguientes:

- De carácter
- Numéricas
- De fecha
- De conversión
- Generales
- Analíticas

3.1.3. Funciones de Varias Filas

Las funciones pueden manipular grupos de filas para dar un resultado por grupo de filas. Estas funciones se conocen también como funciones de grupo (y se describen en una lección posterior).

Nota: Para obtener más información y una lista completa de las funciones disponibles y su sintaxis, consulte Oracle SQL Reference.

3.2. Funciones de una Sola Fila

Las funciones de una sola fila se utilizan para manipular elementos de datos. Aceptan uno o más argumentos y devuelven un valor para cada fila que devuelve la consulta. Un argumento puede ser uno de los siguientes:

- Constante proporcionada por el usuario
- Valor de variable
- Nombre de columna
- Expresión

Las características de las funciones de una sola fila son:

- Actúan en cada fila que se devuelve en la consulta
- Devuelven un resultado por fila
- Posiblemente devuelven un valor de datos de un tipo diferente a aquel al que se
- hace referencia
- Posiblemente esperan uno o más argumentos
- Se pueden utilizar en cláusulas SELECT, WHERE y ORDER BY; se pueden anidar

En la sintaxis:

- function_name: es el nombre de la función
- arg1, arg2: es cualquier argumento que vaya a utilizar la función.

Se puede representar con un nombre de columna o una expresión.

function_name [(arg1, arg2,...)]

Funciones de carácter: Aceptan la entrada de caracteres y pueden devolver valores de carácter y numéricos

Funciones numéricas: Aceptan la entrada de números y devuelven valores numéricos

Funciones de fecha: Operan en valores del tipo de datos DATE (todas las funciones de fecha devuelven un valor del tipo de datos DATE excepto la función MONTHS_BETWEEN, que devuelve un número.) o ASCII (devuelve el valor del carácter)

Funciones Analíticas que generan un resultado a partir del análisis de una serie de resultados

Funciones de conversión: Convierten un valor de un tipo de datos en otro.

Funciones Generales:

- NVL
- NVL2
- NULLIF

- COALESCE
- CASE
- DECODE

3.3. Funciones de Carácter

Las funciones de carácter de una sola fila aceptan datos de caracteres como entrada y pueden devolver valores de carácter y numéricos. Las funciones de carácter se pueden dividir en:

- Funciones de manipulación de mayúsculas/minúsculas
- Funciones de manipulación de caracteres

Función	Objetivo
LOWER(column|expression)	Convierte los valores de carácter alfabéticos a minúsculas
UPPER(column|expression)	Convierte los valores de carácter alfabéticos a mayúsculas
INITCAP(column|expression)	Convierte los valores de carácter alfabéticos a mayúsculas para la primera letra de cada palabra; el resto en minúsculas
CONCAT(column1|expression1, column2|expression2)	Concatena el primer valor de carácter con el segundo; equivalente al operador de concatenación (||)
SUBSTR(column|expression, m[,n])	Devuelve los caracteres especificados del calor de carácter empezando por la posición de carácter m, con n caracteres de longitud (Si m es negativo, el recuento se inicia desde el final del valor de carácter. Si se omite n, se devuelven todos los caracteres hasta el final de la cadena.)
LENGTH(column|expression)	Devuelve el número de caracteres de la expresión Si la expresión contiene NULL devuelve NULL
INSTR(column|expression, 'string', [,m], [n]	Devuelve la posición numérica de una cadena especificada. Opcionalmente, puede proporcionar una posición m para iniciar la búsqueda y la incidencia n de la cadena. m y n tienen por defecto el valor 1, lo que significa iniciar la búsqueda al principio de la búsqueda e informar de la primera incidencia.
LPAD(column|expression, n, 'string')	Rellena el valor de carácter justificado a la derecha hasta un ancho total de n posiciones de carácter.

RPAD(column|expression, n, 'string')	Rellena el valor de carácter justificado a la izquierda hasta un ancho total de n posiciones de carácter
TRIM(leading|trailing|both, trim_character FROM trim_source)	Le permite recortar caracteres de cabecera o finales (o ambos) de una cadena de caracteres. Si trim_character o trim_source son literales de carácter, los debe poner entre comillas simples. Esto está disponible en Oracle8i y versiones posteriores.
REPLACE(text,search_string, replacement_string)	Busca en una expresión de texto una cadena de caracteres y, de encontrarla, la sustituye por una cadena de sustitución especificada

3.3.1. Funciones de Manipulación de Caracteres

Ejemplos de las funciones de manipulación de carácter

```
SELECT 'The job id for '|| UPPER(last_name)||' is '
||LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM employees;
SELECT employee_id, CONCAT(first_name, last_name) NAME,
LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM employees
WHERE SUBSTR(last_name, -1, 1) = 'n';
```

3.3.2. Funciones Numéricas

Las funciones numéricas aceptan la entrada de números y devuelven valores numéricos. Esta sección describe algunas de las funciones numéricas.

3.3.3. Función ROUND

La función ROUND redondea la columna, la expresión o el valor a n posiciones decimales.

Si el segundo argumento es 0 o falta, el valor se redondea a cero posiciones decimales.

Si el segundo argumento es 2, el valor se redondea a dos posiciones decimales.

A la inversa, si el segundo argumento es -2, el valor se redondea a dos posiciones decimales a la izquierda (redondeando a la unidad más cercana a 10).

La función ROUND también se puede utilizar con funciones de fecha.

3.4. Tabla DUAL

La tabla DUAL es propiedad del usuario SYS y pueden acceder a ella todos los usuarios.

Contiene una columna, DUMMY, y una fila con el valor X. La tabla DUAL resulta útil si desea devolver un valor sólo una vez (por ejemplo, el valor de una constante, una pseudocolumna o una expresión que no se deriva de una tabla con datos de usuario). La tabla DUAL se utiliza generalmente para la integridad de sintaxis de la cláusula SELECT, ya que las sentencias

SELECT y FROM son obligatorias y varios cálculos no necesitan seleccionar de tablas reales.

```
SELECT ROUND (45 . 922 , 2 ) , ROUND ( 45 .922 , 0 ) ,
ROUND(45.922,-1)
FROM        DUAL;
```

3.4.1. Función TRUNC

La función TRUNC trunca la columna, la expresión o el valor a n posiciones decimales.

La función TRUNC trabaja con argumentos parecidos a los de la función ROUND. Si el

segundo argumento es 0 o falta, el valor se trunca a cero posiciones decimales.

Si el segundo argumento es 2, el valor se trunca a dos posiciones decimales.

A la inversa, si el segundo argumento es -2, el valor se trunca a dos posiciones decimales a la izquierda.

Si el segundo argumento es -1, el valor se trunca a una posición decimal a la izquierda.

Como la función ROUND, la función TRUNC también se puede utilizar con funciones de fecha.

```
SELECT TRUNC ( 45 .922 ,2 ) , TRUNC  ( 45 .922 ) ,
TRUNC (45.922,-1)
FROM          DUAL;
```

3.4.2. Función MOD

La función MOD busca el resto del primer argumento dividido por el segundo argumento. El ejemplo calcula el resto del salario después de dividirlo por 5.000 para todos los empleados cuyo identificador de puesto es SA_REP.

Nota: La función MOD se utiliza a menudo para determinar si un valor es par o impar.

```
SELECT last_name, salary, MOD(salary, 5000)
FROM          employees
WHERE      job_id = 'SA_REP';
```

3.5. Formato de Fecha de Oracle

La base de datos Oracle almacena las fechas en un formato numérico interno, que representa el siglo, el año, el mes, el día, las horas, los minutos y los segundos.

El formato de visualización y va a depender de los parámetros de creación de la base de datos. También de ello depende como se muestre el nombre del mes y de los días

Las fechas válidas de Oracle van del 1 de enero de 4712 a.C. al 21 de diciembre de 9999 d.C.

En el ejemplo, la salida de la columna HIRE_DATE se muestra en el formato por defecto DD-MON-YYYY.

Sin embargo, las fechas no se almacenan en la base de datos en este formato. Se almacenan todos los componentes de fecha y hora. Así pues, aunque una fecha HIRE_DATE como 17-JUN-87 se muestra como día, mes y año, también hay información de hora y de siglo asociada a la fecha. Los datos completos podrían ser 17 de junio de 1987, 5:10:42 p.m.

Esta fecha se almacena internamente así:

SIGLO AÑO MES DÍA HORA MINUTO SEGUNDO

```
19        87      06    17      17 10         42
```

3.5.1. Función SYSDATE

SYSDATE es una función de fecha que devuelve la fecha y la hora actuales del servidor de bases de datos. Puede utilizar SYSDATE igual que cualquier otro nombre de columna. Por ejemplo, puede mostrar la fecha actual seleccionando SYSDATE en una tabla. Se suele seleccionar SYSDATE en una tabla ficticia denominada DUAL.

```
SELECT SYSDATE
FROM      DUAL;
```

3.5.2. Aritmética de Fechas

Como la base de datos almacena fechas como números, puede realizar cálculos mediante operadores aritméticos como la suma o la resta. Puede sumar y restar constantes numéricas además de fechas.

Puede utilizar las siguientes operaciones:

Operación	Resultado	Descripción
fecha + número	Fecha	Suma un número de días a una fecha
fecha - número	Fecha	Resta un número de días a una fecha
fecha - fecha	Número de días:	Resta una fecha a otra
fecha + número/24	Fecha	Suma un número de horas a una fecha

El ejemplo muestra el apellido y el número de semanas de empleo para todos los empleados del departamento 90.

Resta la fecha en la que se contrató al empleado de la fecha actual (SYSDATE) y divide el resultado por 7 para calcular el número de semanas que un trabajador lleva empleado.

Si se resta una fecha más actual a una más antigua, la diferencia será un número negativo

3.5.3. Funciones de Fecha

Las funciones de fechas operan en fechas de Oracle. Todas las funciones de fecha devuelven un valor del tipo de datos DATE excepto MONTHS_BETWEEN, que devuelve un valor numérico.

Función	Resultado
MONTHS_BETWEEN(date1, date2)	Número de meses entre dos fechas
ADD_MONTHS(date, n)	Agrega meses de calendario a una fecha
NEXT_DAY(date, 'char')	Día siguiente a la fecha especificada
LAST_DAY(date)	Último día del mes
ROUND(date[, 'fmt'])	Redondea la fecha
TRUNC(date[, 'fmt'])	Trunca la fecha

-

- MONTHS_BETWEEN(date1, date2): Busca el número de meses entre date1 y date2. El resultado puede ser positivo o negativo.

Si date1 es posterior a date2, el resultado es positivo; si date1 es anterior a date2, el resultado es negativo. La parte no entera del resultado representa una porción del mes.

- ADD_MONTHS(date, n): Agrega un número n de meses de calendario a date. El valor de n debe ser un entero y puede ser negativo.

- NEXT_DAY(date, 'char'): Busca la fecha del siguiente día de la semana especificado ('char') después de date. El valor de char puede ser un número que represente un día o una cadena de caracteres.

• LAST_DAY(date): Busca la fecha del último día del mes que contiene date

- ROUND(date[,'fmt']): Devuelve date redondeado a la unidad especificada por el modelo de formato fmt. Si se omite el modelo de formato fmt, date se redondeará al día más cercano.

- TRUNC(date[, 'fmt']): Devuelve date con la porción de tiempo del día truncada a la unidad especificada por el modelo de formato fmt. Si se omite el modelo de formato fmt, date se trunca al día más cercano.

Por ejemplo, muestre el número de empleado, la fecha de contratación, el número de meses de empleo, la fecha de revisión semestral, el primer viernes tras la fecha de contratación y el mes de contratación de todos los empleados que lleven contratados menos de 26 meses.

```
SELECT employee_id, hire_date,
    MONTHS_BETWEEN (SYSDATE, hire_date) CONTRATACION,
    ADD_MONTHS (hire_date, 6) REVISION,
    NEXT_DAY (hire_date, 'FRIDAY'),
    LAST_DAY(hire_date)
FROM        employees
WHERE       MONTHS_BETWEEN (SYSDATE, hire_date) < 26;
```

De acuerdo a cual sea el idioma con el que fue creada la base de datos el nombre del día puede variar. Si la base de datos esta en castellano corresponderá 'VIERNES' en lugar de 'FRIDAY'

3.6. Funciones de Conversión

Además de los tipos de datos Oracle, las columnas de tablas de una base de datos Oracle se pueden definir mediante tipos de datos ANSI, DB2 y SQL/DS. Sin embargo Oracle Server convierte internamente esos tipos de datos a tipos de datos Oracle.

En algunos casos, Oracle Server utiliza datos de un tipo de datos donde espera datos de un tipo de datos diferente. Cuando sucede esto, Oracle Server puede convertir automáticamente los datos al tipo de datos esperado. Esta conversión de tipos de datos lo puede realizar implícitamente Oracle Server, o explícitamente el usuario.

Las conversiones de tipos de datos explícitas se realizan mediante las funciones de conversión. Las funciones de conversión convierten un valor de un tipo de dato a otro.

Generalmente, la forma de los nombres de función utiliza la convención data type TO data type. El primer tipo de datos es el de entrada; el segundo, el de salida.

Nota: Aunque está disponible la conversión de tipos de datos implícita, se recomienda que realice una conversión de tipos de datos explícita para asegurar la fiabilidad de las sentencias SQL.

3.6.1. Conversión de Tipos de Datos Implícita

La asignación es correcta si Oracle Server puede convertir el tipo de datos del valor utilizado en la asignación al de destino.

Por ejemplo, la expresión hire_date > '01-JAN-90' da como resultado la conversión implícita de la cadena '01-JAN-90' a una fecha.

En general, Oracle Server utiliza la regla para expresiones cuando se necesita una conversión de tipo de datos en lugares no cubiertos por una regla para conversiones de asignación.

Por ejemplo, la expresión salary = '20000' da como resultado la conversión implícita de la cadena '20000' al número 20000.

Nota: Las conversiones de CHAR a NUMBER sólo son correctas si la cadena de caracteres representa un número válido

3.6.2. Conversión de Tipos de Datos Explícita

SQL proporciona tres funciones para convertir un valor de un tipo de datos a otro.

Función	Objetivo
TO_CHAR(number|date,[fmt], [nlsparams])	Convierte un valor numérico o de fecha a una cadena de caracteres VARCHAR2 con el modelo de formato fmt Conversión numérica: El parámetro nlsparams especifica los siguientes caracteres, que son devueltos por elementos de formato numérico: • Carácter decimal • Separador de grupos • Símbolo de divisa local • Símbolo de divisa internacional Si se omite nlsparams o cualquier otro parámetro, esta función utiliza los valores de parámetros por defecto para la sesión. Conversión de fecha: El parámetro nlsparams especifica el lenguaje en que se devolverán los nombres y las abreviaturas de mes y de día. Si se omite este parámetro, esta función utiliza los lenguajes de fecha por defecto para la sesión.
TO_NUMBER(char,[fmt], [nlsparams])	Convierte una cadena de caracteres que contenga dígitos en un número con el formato especificado por el modelo de formato opcional fmt. El parámetro nlsparams tiene el mismo objetivo en esta función que en la función TO_CHAR de conversión numérica.
TO_DATE(char,[fmt], [nlsparams])	Convierte una cadena de caracteres que representa una fecha en un valor de fecha de acuerdo con el fmt que se haya especificado. Si se omite fmt, el formato es DD-MON-YY. El parámetro nlsparams tiene el mismo objetivo en esta función que en la función TO_CHAR de conversión de fecha.

3.6.3. Uso de la Función TO_CHAR con Fechas

Anteriormente, todos los valores de datos de Oracle se mostraban en formato DD-MON-YY. Puede utilizar la función TO_CHAR para convertir una fecha de este formato por defecto al que especifique.

Instrucciones

- El modelo de formato debe ir entre comillas simples y es sensible a mayúsculas/minúsculas.
- El modelo de formato puede incluir cualquier elemento de formato de fecha. Asegúrese de separar el valor de fecha del modelo de formato con una coma.
- Los nombres de días y meses de la salida se rellenan automáticamente con espacios en blanco.
- Para eliminar espacios en blanco rellenados o para suprimir ceros iniciales, utilice el elemento fm del modo de relleno.

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM   employees
WHERE  last_name = 'Higgins';
```

3.6.4. Elementos del Modelo de Formato de Fecha

Elemento	Resultado
YYYY	Año completo con números
YEAR	Nombre completo de año con letras (en inglés)
MM	Valor de dos dígitos para el mes
MONTH	Nombre completo del mes
MON	Abreviatura de tres letras del mes
DY	Abreviatura de tres letras del día de la semana
DAY	Nombre completo del día de la semana

DD	Día del mes con números
YYY o YY o Y	Los últimos tres, dos o uno dígitos del año
Y,YYY	Año con una coma en esa posición
IYYY, IYY, IY, I	Año de cuatro, tres, dos o un dígitos basado en el estándar ISO
SYEAR o YEAR	Nombre completo de año con letras; Oracle Server agrega - a las fechas a. C.
Q	Trimestre
MM	Mes: valor de dos dígitos
MONTH	Nombre del mes rellenado con espacios en blanco hasta la longitud de nueve caracteres.
MON	Nombre del mes, abreviatura de tres letras
WW o W	Semana del año o del mes
DDD o DD o D	Día del año, del mes o de la semana
DAY	Nombre del día rellenado con espacios en blanco hasta la longitud de nueve caracteres
DY	Nombre del día, abreviatura de tres letras

3.6.5. Elemento de Formato de Fecha: Formatos de Hora

Utilice los formatos que se muestran en las tablas siguientes para mostrar información de hora y literales y para cambiar números en cifras a números en letras

Elemento	Resultado
AM o PM	Indicador de meridiano
A.M. o P.M.	Indicador de meridiano con puntos
HH o HH12 o HH24	Hora del día, u hora (1-12) u hora (0-22)
MI	Minuto (0-59)
SS	Segundo (0-59)
SSSSS	Segundos desde la medianoche (0-86299)

Otros formatos:

Elemento	Descripción
/ ,	La puntuación se reproduce en el resultado.
"de"	Las comillas se reproducen en el resultado

Especificación de Sufijos para Influir en la Visualización de Números

Elemento	Descripción
TH	Número ordinal (por ejemplo, DDTH para 4TH)
SP	Número completo con letras (por ejemplo, DDSP para FOUR)
SPTH o THSP	Número ordinal completo con letras (por ejemplo, para DSPTH FOURTH)

3.6.6. Uso de la Función TO_CHAR con Fechas

La sentencia SQL muestra los apellidos y las fechas de contratación de todos los empleados. La fecha de contratación aparece como 17 Junio 1987.

Ejemplo

```
SELECT      last_name,
TO_CHAR(hire_date,
' fmDdsp th "o f " Mon t h YYYY f mHH :M I : SS AM ' ) H IREDATE
```

```
FROM        employees;
```

3.6.7. Uso de la Función TO_CHAR con Números

Al trabajar con valores numéricos como cadenas de caracteres, debe convertir esos números al tipo de datos de carácter mediante la función TO_CHAR, que traduce un valor del tipo de datos NUMBER al tipo de datos VARCHAR2. Esta técnica es especialmente útil con la concatenación.

Sintaxis

```
TO_CHAR(number, 'format_model') ddspth
```

Elementos de Formato Numérico

Si está convirtiendo un número al tipo de datos de carácter, puede utilizar estos elementos de formato:

Elemento	Descripción	Ejemplo	Resultado
9	Posición numérica (el número de nueves determina el ancho de la visualización)	999999	12245
0	Visualización de ceros iniciales	099999	012245
$	Signo de dólar flotante	$999999	$12245
L	Símbolo de divisa local flotante	L999999	$12245
D	Devuelve el carácter decimal en la posición especificada. El valor por defecto es un punto (.).	99D99	12.24
G	Devuelve el separador de grupos en la posición especificada. Puede especificar varios separadores de grupo en un modelo de formato numérico.	9,999	1,224
,	Coma en la posición especificada	999,999	122,456
MI	Signo menos a la derecha (valores negativos)	999999MI	122456-
PR	Números negativos entre corchetes	999999PR	<1224>
EEEE	Notación científica (el formato debe especificar cuatro letras E)	99,999EEEE	1,224E+02
U	Devuelve la divisa dual "euro" (u otra) en la posición	U9999	€1224

	especificada		
V	Multiplica por 10 n veces (n = número de nueves tras V)	9999V99	122400
S	Devuelve el valor negativo o positivo	S9999	+1224
B	Visualiza los valores cero como espacios en blanco, no como 0	B9999,99	1224,00

- Oracle Server muestra una cadena de signos numéricos (#) en lugar de un número completo cuyos dígitos excedan el número de dígitos que se proporciona en el modelo de formato.
- Oracle Server redondea el valor decimal almacenado al número de posiciones decimales que se proporciona en el modelo de formato.

3.6.8. Uso de las Funciones TO_NUMBER y TO_DATE

En ocasiones, deberá convertir una cadena de caracteres a número o a fecha. Para ello, utilice las funciones TO_NUMBER o TO_DATE. El modelo de formato que elija se basará en los elementos de formato demostrados anteriormente.

- El modificador fx especifica la correspondencia exacta del argumento de carácter y el modelo de formato de fecha de una función TO_DATE.
- La puntuación y el texto entre comillas del argumento de carácter debe corresponder exactamente (excepto en las mayúsculas/minúsculas) con las partes correspondientes del modelo de formato.
- El argumento de carácter no puede contener espacios en blanco adicionales. Sin fx, Oracle ignora los espacios en blanco adicionales.
- Los datos numéricos del argumento de carácter debe tener el mismo número de dígitos que el elemento correspondiente del modelo de formato. Sin fx, los números del argumento de carácter pueden omitir los ceros iniciales.

Sintaxis

```
TO_NUMBER(char[, 'format_model'])
TO_DATE(char[, 'format_model'])
```

3.6.9. Elemento de Formato de Fecha RR

El formato de fecha RR es parecido al elemento YY, pero lo puede utilizar para especificar siglos diferentes. Utilice el elemento de formato RR en lugar de YY para que el siglo del valor de retorno varíe según el año de dos dígitos especificado y los dos dígitos del año actual. La tabla de la diapositiva resume el comportamiento del elemento RR Anidamiento de Funciones

Las funciones de una sola fila se pueden anidar hasta cualquier profundidad. Las funciones anidadas se evalúan desde el nivel más interno al más externo. A continuación, se ofrecen varios ejemplos de la flexibilidad de estas funciones.

```
SELECT l as t _name ,
UPPER (CONCAT ( SUBSTR ( LAST_NAME , 1 , 8 ) , '_US ' ))  FROM
emp loyees
WHERE      depa r tmen t_ id = 60 ;
```

El ejemplo muestra los apellidos de los empleados del departamento 60. La evaluación de la sentencia SQL implica tres pasos:

1. La función interna recupera los primeros ocho caracteres del apellido.

Result1 = SUBSTR (LAST_NAME, 1, 8)

2. La función externa concatena el resultado on _US.

Result2 = CONCAT(Result1, '_US')

2. La función externa convierte los resultados a mayúsculas.

Toda la expresión se convierte en la cabecera de la columna porque no se ha proporcionado alias de columna.

3.6.10. Funciones Generales

Estas funciones pueden utilizar cualquier tipo de datos y están relacionados con el uso de valores nulos en la lista de expresiones.

Función	Descripción
NVL	Convierte un valor nulo en un valor real
NVL2	Si expr1 no es nulo, NVL2 devuelve expr2. Si expr1 es nulo, NVL2 devuelve expr2. El argumento expr1 puede tener cualquier tipo de datos.
NULLIF	Compara dos expresiones y devuelve un valor nulo si son iguales; devuelve la primera expresión si no son iguales
COALESCE	Devuelve la primera expresión no nula de la lista de expresiones

3.7. Función NVL

Para convertir un valor nulo en un valor real, utilice la función NVL. Sintaxis

NVL (exp r 1 , exp r2)

En la sintaxis:

- expr1 es el valor o la expresión de origen que puede contener un valor nulo
- expr2 es el valor de destino para convertir el valor nulo

Puede utilizar la función NVL para convertir cualquier tipo de datos, pero el valor de retorno es siempre el mismo que el tipo de datos de expr1.

Conversiones NVL para Varios Tipos de Datos

3.7.1. Uso de la Función NVL

Para calcular la compensación anual de todos los empleados, debe multiplicar el salario mensual por 12 y sumar el porcentaje de comisión al resultado.

```
SELECT las t_name , sa la ry , comm iss i on_pc t,
(sa la r y *12 ) + (sa la r y* 12 *comm i ss ion_pc t ) AN_SAL FROM
emp loyees ;
```

3.7.2. Uso de la Función NVL2

La función NVL2 examina la primera expresión. Si la primera expresión no es nula, la función NVL2 devuelve la segunda expresión. Si la primera expresión es nula, se devuelve la tercera expresión.

Sintaxis

NVL2(expr1, expr2, expr2)

En la sintaxis:

- expr1 es el valor o la expresión de origen que puede contener un valor nulo
- expr2 es el valor que se devuelve si expr1 no es nulo
- expr2 es el valor que se devuelve si expr2 es nulo

En el ejemplo que se muestra, se examina la columna COMMISSION_PCT. Si se detecta un valor, se devuelve la segunda expresión de SAL+COMM. Si la columna COMMISSION_PCT contiene un valor nulo, se devuelve la tercera expresión de SAL. El argumento expr1 puede tener cualquier tipo de datos. Los argumentos expr2 y expr2 pueden tener cualquier tipo de datos excepto LONG. Si los tipos de datos de expr2 y expr2 son diferentes, Oracle Server convierte expr2 al tipo de datos de expr2 antes de compararlos a menos que expr2 sea una constante nula. En el último caso, no es necesaria una conversión del tipo de datos. El tipo de datos del valor de retorno es siempre el mismo que el tipo de datos de expr2, a menos que expr2 sean datos de carácter, en cuyo caso el tipo de datos del valor de retorno es VARCHAR2.

```
SELECT last_name,      salary, commission_pct,
              NVL2(commission_pct,
'SAL+COMM',  'SAL') income
FROM      employees WHERE department_id IN (50, 80);
```

3.7.3. Uso de la Función NULLIF

La función NULLIF compara dos expresiones. Si son iguales, la función devuelve un valor nulo. Si no son iguales, la función devuelve la primera expresión. No puede especificar el literal NULL para la primera expresión.

Sintaxis

NULLIF (expr1, expr2)

En la sintaxis:

- expr1 es el valor de origen que se compara con expr2
- expr2 es el valor de origen que se compara con expr1 (Si no es igual que expr1, se devuelve expr1.)

En el ejemplo que se muestra, la longitud del nombre de la tabla EMPLOYEES se compara con el apellido de la tabla EMPLOYEES. Si las longitudes del nombre y el apellido son iguales, se devuelve un valor nulo. Si las longitudes del nombre y el apellido no son iguales, se muestra la longitud del nombre.

Nota: La función NULLIF es lógicamente equivalente a esta expresión CASE. La expresión CASE se analiza en ítem posterior: CASE WHEN expr1 = expr 2 THEN NULL ELSE expr1 END

3.7.4. Uso de la Función COALESCE

La función COALESCE devuelve la primera expresión no nula de la lista.

Sintaxis

COALESCE (expr1, expr2, ... exprn)

En la sintaxis:

- expr1 devuelve esta expresión si no es nula

- expr2 devuelve esta expresión si la primera expresión es nula y esta expresión no lo es

- exprn devuelve esta expresión si las expresiones precedentes son nulas

Todas las expresiones deben ser del mismo tipo de datos.

En el ejemplo, si el valor MANAGER_ID no es nulo, se muestra. Si el valor MANAGER_ID es nulo, se muestra COMMISSION_PCT. Si los valores MANAGER_ID y COMMISSION_PCT son nulos, se muestra el valor -1.

```
SELECT last_name, COALESCE(manager_id,commission_pct, -1)
comm
FROM        employees
ORDER BY commission_pct;
```

3.7.5. Expresiones Condicionales

Los dos métodos utilizados para implementar procesamiento condicional (lógica IF-THEN-ELSE) en una sentencia SQL son la expresión CASE y la función DECODE.

Nota: La expresión CASE cumple con ANSI SQL. La función DECODE es específica de la sintaxis Oracle.

3.7.6. Expresión CASE

Las expresiones CASE le permiten utilizar la lógica IF-THEN-ELSE en sentencias SQL sin llamar a procedimientos.

En una expresión CASE simple, Oracle Server busca el primer par WHEN ... THEN en el que expr sea igual a comparison_expr y devuelve return_expr. Si ninguno de los pares WHEN ... THEN cumplen esta condición y si existe una cláusula ELSE, Oracle Server devuelve else_expr.

De lo contrario, Oracle Server devuelve un valor nulo. No puede especificar el literal NULL para todas las expresiones return_exprs y else_expr.

Todas las expresiones (expr, comparison_expr y return_expr) deben ser del mismo tipo de datos, que puede ser CHAR, VARCHAR2, NCHAR o NVARCHAR2.

3.7.7. Uso de la Expresión CASE

En la sentencia SQL de la diapositiva, se descodifica el valor de JOB_ID. Si JOB_ID es IT_PROG, el aumento de salario es del 10 %; si JOB_ID es ST_CLERK, el aumento de salario es del 15 %; si JOB_ID es SA_REP, el aumento de salario es del 20 %. Para el resto de roles de trabajo, no hay aumento de salario.

Se puede escribir la misma sentencia con la función DECODE.

Esto es un ejemplo de expresión CASE buscada. En una expresión CASE buscada, la búsqueda se produce de izquierda a derecha hasta que se encuentre una incidencia de la ondición mostrada y, entonces, devuelve la expresión de retorno. Si no se encuentra ninguna condición que sea verdadera y si existe una cláusula ELSE, se devuelve la expresión de retorno de la cláusula ELSE; de lo contrario, se devuelve NULL.

```
SELECT l as t _name ,sa la ry ,
 (CASE WHEN sa l a r y<5000 THEN ' Ba j o '
WHEN sa l a ry<10000 THEN 'Med i o '
WHEN sa l a ry<20000 THEN 'Bueno '
ELSE 'Exce l en te '
END) ‘Ca l i f icac ion  sa la r i a l ’
FROM emp l oyees ;
```

3.8. Función DECODE

La función DECODE descodifica una expresión de forma parecida a la lógica IF-THEN-ELSE que se utiliza en varios lenguajes. La función DECODE descodifica expression tras compararla con cada valor search. Si la expresión es igual que search, se devuelve result.

Si se omite el valor por defecto, se devuelve un valor nulo donde un valor de búsqueda no corresponda a ninguno de los valores del resultado.

3.8.1. Uso de la Función DECODE

En la sentencia SQL, se prueba el valor de JOB_ID. Si JOB_ID es IT_PROG, el aumento de salario es del 10 %; si JOB_ID es ST_CLERK, el aumento de salario es del 15 %; si JOB_ID es SA_REP, el aumento de salario es del 20 %. Para el resto de roles de trabajo, no hay aumento de salario.

```
SELECT last_name, job_id, salary
DECODE(job_id, 'IT_PROG', 1.10*salary,
    'ST_CLERK', 1.15*salary,
    'SA_REP',       1.20*salary,
salary)
Revision_Salarial
FROM employees;
```

Ejercicios

1. Escriba una consulta para mostrar la fecha actual. Etiquete la columna como Date

2. El departamento de recursos humanos necesita mostrar el número de empleado, el apellido, el salario y el salario aumentado en un 15,5 % (expresado como número entero) de cada empleado. Etiquete la columna como Nuevo Salario. Guarde la sentencia SQL en un archivo de texto denominado lab_02_02.sql.

3. Ejecute la consulta del archivo lab_02_02.sql.

4. Modifique la consulta lab_02_02.sql para agregar una columna que reste el antiguo salario al nuevo salario. Etiquete la columna como Increase. Guarde el contenido en un archivo denominado lab_02_04.sql. Ejecute la consulta revisada.

5. Cree una consulta que muestre los apellidos y los importes de comisión de los empleados. Si un empleado no gana ninguna comisión, muestre "No Commission". Etiquete la columna como COMM.

6. Mediante la función DECODE, escriba una consulta que muestre el grado de todos los empleados basándose en el valor de la columna JOB_ID, mediante estos datos:

Puesto	Grado
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
Ninguno de las anteriores	0

4. Trabajo con varias Tablas

4.1. Objetivos

Esta lección explica cómo obtener datos de más de una tabla. Una unión se utiliza para ver información de varias tablas. De esta forma, puede unir tablas para ver información de más de una tabla

4.1.1. Obtención de Datos de Varias Tablas

En ocasiones, tiene que utilizar datos de más de una tabla.

Si tenemos que ubicar empleados que pertenezcan a un departamento identificando el nombre del departamento debemos proceder de la siguiente forma:

- Los identificadores de empleado se encuentran en la tabla EMPLOYEES.
- Los identificadores de departamento se encuentran en las tablas EMPLOYEES y DEPARTMENTS.
- Los nombres de departamento se encuentran en la tabla DEPARTMENTS.

Para crear el informe, debe enlazar las tablas EMPLOYEES y DEPARTMENTS y acceder a datos de ambas

4.1.2. Tipos de Uniones

Para unir tablas, puede utilizar la sintaxis de unión compatible con el estándar SQL:1999.

Nota: antes de la versión Oracle9i, la sintaxis de unión era diferente a los estándares ANSI.

La sintaxis de unión compatible con SQL:1999 no ofrece ventajas de rendimiento con

respecto a la sintaxis de unión propietaria de Oracle que existía en las versiones anteriores.

Las uniones compatibles con el estándar SQL:1999 son:

- Uniones cruzadas
- Uniones naturales
- Cláusula USING
- Uniones externas completas (o de dos lados)
- Condiciones de unión arbitrarias para uniones Externas

4.1.3. Definición de Uniones

En la sintaxis

```
SELECT      table1.column, table2.column
FROM        table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] | [JOIN table2
ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
ON (table1.column_name = table2.column_name)]| [CROSS JOIN
table2];
```

- table1.column muestra la tabla y la columna de las que se recuperan los datos
- NATURAL JOIN une dos tablas basándose en el mismo nombre de columna
- JOIN table USING column_name realiza una unión igualitaria basándose en el nombre de columna
- JOIN table ON table1.column_name realiza una unión igualitaria basándose en la condición de la cláusula ON, = table2.column_name
- LEFT/RIGHT/FULL OUTER se utiliza para realizar uniones externas
- CROSS JOIN devuelve un producto cartesiano de las dos tablas

4.1.4. Creación de Uniones Naturales

Puede unir tablas automáticamente basándose en columnas de las dos tablas que tengan tipos de datos y nombres correspondientes. Hágalo mediante las palabras clave NATURAL JOIN.

Nota: la unión sólo se puede realizar en las columnas que tengan los nombres y los tipos de datos iguales en ambas tablas. Si las columnas tienen el mismo nombre pero diferentes tipos de datos, la sintaxis de NATURAL JOIN producirá un error.

```
SELECT department_id, department_name,
location_id, city
FROM departments
NATURAL JOIN locations ;
```

4.1.5. Creación de Uniones con la Cláusula USING

Las uniones naturales pueden utilizar todas las columnas con nombres y tipos de datos correspondientes para unir las tablas. Se puede utilizar la cláusula USING para especificar únicamente las columnas que se deben utilizar para una unión igualitaria. Las columnas a las que se hace referencia en la cláusula USING no deben tener cualificador (nombre de tabla o alias) en ninguna parte de la sentencia SQL.

Por ejemplo, esta sentencia es válida:

```
SELECT l.city, d.department_name
FROM locations l JOIN departments d
USING (location_id)
WHERE location_id = 1400;
La sentencia siguiente no es válida porque la cláusula WHERE
cualifica a LOCATION_ID:
SELECT l.city, d.department_name
FROM locations l JOIN departments d
  USING (location_id)
WHERE d.location_id = 1400;
ORA-23134: column part of USING clause cannot have qualifier
```

Se aplica la misma restricción a las uniones naturales. Por tanto, las columnas con el mismo nombre en ambas tablas se deben utilizar sin cualificadores.

4.1.6. Recuperación de Registros con la Cláusula USING

Para determinar el nombre de departamento de un empleado, compare el valor de la columna DEPARTMENT_ID de la tabla EMPLOYEES con los valores de DEPARTMENT_ID de la tabla DEPARTMENTS. La relación entre las tablas EMPLOYEES y DEPARTMENTS es una unión igualitaria (es decir, los valores de la columna DEPARTMENT_ID de ambas tablas deben ser iguales).

Con frecuencia, este tipo de unión implica complementos de clave primaria y ajena.

Nota: Las uniones igualitarias se denominan también uniones simples o uniones internas.

```
SELECT employees.employee_id, employees.last_name,
       departments.location_id, department_id
FROM   employees
JOIN departments
                    USING (department_id) ;
```

4.2. Calificación de Nombres de Columna Ambiguos

Debe cualificar los nombres de las columnas con el nombre de la tabla para evitar ambigüedades.

Sin los prefijos de tabla, la columna DEPARTMENT_ID en la lista SELECT podría ser de la tabla DEPARTMENTS o de la tabla EMPLOYEES.

Resulta necesario agregar el prefijo de tabla para ejecutar la consulta:

```
SELECT employees.employee_id, employees.last_name,
departments.department_id, epartments.location_id
FROM   employees
    JOIN departments
        ON employees.department_id =
departments.department_id;
```

Si no existen nombres de columna comunes en las dos tablas, no es necesario cualificar las columnas. Sin embargo, utilizar el prefijo de tabla mejora el rendimiento, ya que le indica a Oracle Server exactamente dónde puede encontrar las columnas.

Nota: Al realizar uniones mediante la cláusula USING, no puede cualificar una columna que se utilice en la propia cláusula USING. Lo que es más, si se utiliza esa columna en cualquier otra parte de la sentencia SQL, no puede aplicarle un alias.

4.2.1. Uso de Alias de Tabla

Cualificar nombres de columna con nombres de tabla puede llevar mucho tiempo, especialmente si los nombres de tabla son largos. Puede utilizar alias de tabla en lugar de nombres de tabla.

Al igual que los alias de columna cambian el nombre de las columnas, los alias de tabla hacen lo propio con las tablas. Los alias de tabla ayudan a reducir el código SQL, con lo que se consume menos memoria.

Observe cómo se identifican en el ejemplo los alias de tabla de la cláusula FROM.

El nombre de tabla se especifica entero, seguido de un espacio y, a continuación, el alias de tabla. Se le ha dado a la tabla EMPLOYEES el alias e y a la tabla DEPARTMENTS, el alias d.

Instrucciones

- Los alias de tabla pueden contener hasta 30 caracteres, pero son mejores los cortos que los largos.
- Si se utiliza un alias de tabla para un nombre de tabla concreto en la cláusula FROM, se debe sustituir el nombre de tabla por alias de tabla en toda la sentencia SELECT.
- Los alias de tabla deben ser significativos.
- Los alias de tabla son válidos únicamente para la sentencia SELECT actual.

```
SELECT e.employee_id, e.last_name,d.location_id,
department_id
FROM   employees e
         JOIN departments d
       USING (department_id);
```

4.2.2. Unión de una Tabla a Sí Misma

A veces, tendrá que unir una tabla a sí misma. Para encontrar el nombre del supervisor de cada empleado, debe unir la tabla EMPLOYEES a sí misma o realizar una autounión. Por ejemplo, para encontrar el nombre del supervisor de Lorentz, debe:

- Buscar Lorentz en la tabla EMPLOYEES en la columna LAST_NAME.
- Buscar el número de supervisor para Lorentz en la columna MANAGER_ID.

El número del supervisor de Lorentz es 103.

Buscar el nombre del supervisor con el valor EMPLOYEE_ID 103 en la columna LAST_NAME. El número de empleado de Hunold es 103, por lo que Hunold es el supervisor de Lorentz.

En este proceso, se consulta la tabla dos veces. La primera vez, se consulta la tabla es para buscar Lorentz en la columna LAST_NAME y el valor MANAGER_ID de 103. La

segunda vez, se consulta la columna EMPLOYEE_ID para buscar 103 y la columna LAST_NAME para buscar Hunold.

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e
JOIN employees m
ON ( e . manager_ id = m .employee_ id ) ;
```

4.2.3. Aplicación de Condiciones Adicionales a una Unión

Puede aplicar condiciones adicionales a la unión.

El ejemplo realiza una unión de las tablas EMPLOYEES y DEPARTMENTS y, además, muestra sólo a los empleados con el identificador de supervisor 149.

Para agregar condiciones adicionales a la cláusula ON, puede agregar cláusulas AND.

De forma alternativa, puede utilizar una cláusula WHERE para aplicar condiciones adicionales.

```
SELECT e.employee_id, e.last_name, e.department_id,
          d.department_id, d.location_id
FROM   employees e
JOIN departments d
ON (e .department_ id = d .department_ id )
AND e .manager_ id = 149 ;
```

4.2.4. Reacción de Uniones en Tres Sentidos

Una unión en tres sentidos es una unión de tres tablas. En la sintaxis compatible con SQL:1999, las uniones se realizan de izquierda a derecha. De esta forma, la primera unión que se debe realizar es EMPLOYEES JOIN DEPARTMENTS. La primera condición de unión puede hacer referencia a columnas de EMPLOYEES y DEPARTMENTS pero no puede hacer referencia a columnas de LOCATIONS. La segunda condición de unión puede hacer referencia a columnas de las tres tablas

```
SELECT employee_id, city, department_name
FROM   employees e
```

JOIN depa r tmen ts d

```
ON  d.department_id = e.department_id
  JOIN          locations l
ON  d.location_id = l.location_id;
```

4.2.5. Uniones Externas

Si una fila no satisface una condición de unión, no aparecerá en el resultado de la consulta. Por ejemplo, en la condición de unión igualitaria de las tablas EMPLOYEES y DEPARTMENTS, encontramos que hay una serie de departamentos que no aparece porque no hay empleados con ese identificador de departamento registrado en la tabla EMPLOYEES.

Para devolver el registro de departamento que no tiene empleados, puede utilizar una unión externa.

4.2.6. Uniones INNER frente a OUTER

Unir tablas con las cláusulas NATURAL JOIN, USING u ON da como resultado una unión interna.

Las filas sin correspondencias no se muestran en la salida. Para devolver las filas sin correspondencias, puede utilizar una unión externa.

Una unión externa devuelve todas las filas que satisfacen la condición de unión y devuelve también algunas o todas las filas de una tabla para las que ninguna fila de la otra tabla satisface la condición de unión.

Hay tres tipos de uniones externas:

- LEFT OUTER
- RIGHT OUTER
- FULL OUTER

4.2.7. LEFT OUTER JOIN

Esta consulta recupera todas las filas de la tabla EMPLOYEES, que es la tabla izquierda aunque no haya correspondencias en la tabla DEPARTMENTS.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
LEFT OUTER JOIN departments d
ON   (e.department_id = d.department_id) ;
```

4.2.8. RIGHT OUTER JOIN

Esta consulta recupera todas las filas de la tabla DEPARTMENTS, que es la tabla derecha aunque no haya correspondencias en la tabla EMPLOYEES.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON (e.department_id = d.department_id)
```

4.2.9. FULL OUTER JOIN

Esta consulta recupera todas las filas de la tabla EMPLOYEES, aunque no haya

correspondencias en la tabla DEPARTMENTS. También recupera todas las filas de la tabla DEPARTMENTS, aunque no haya correspondencias en la tabla EMPLOYEES.

SELECT e.last_name, d.department_id, d.department_name

```
FROM   employees  e
FULL OUTER JOIN departments d
                ON (e.department_id = d.department_id ) ;
```

4.3. Productos Cartesianos

Cuando una condición de unión no es válida o se omite por completo, el resultado es un producto cartesiano, en el que se muestran todas las combinaciones de filas. Todas las filas de la primera tabla se unen a todas las filas de la segunda tabla.

Un producto cartesiano tiende a generar gran cantidad de filas, con lo que el resultado no suele ser de utilidad. Debería incluir siempre una condición de unión válida a menos que tenga la necesidad específica de combinar todas las filas de todas las tablas.

Los productos cartesianos resultan útiles si necesita generar gran número de filas para simular una cantidad aceptable de datos.

4.3.1. Generación de un Producto Cartesiano

Se genera un producto cartesiano si se omite una condición de unión.

Por ejemplo tenemos una consulta que muestra el apellido del empleado y el nombre del departamento de las tablas EMPLOYEES y DEPARTMENTS. Como no se ha especificado ninguna condición de unión, todas las filas (127 filas) de la tabla EMPLOYEES se unen a todas las filas (27 filas) de la tabla DEPARTMENTS, con lo que se generan 2889 registros en la salida.

Un producto cartesiano omite los nulos

```
SELECT e.last_name, e.employee_id, d.department_id,
            d.department_name
FROM    employees e, departments d
```

4.3.2. Reacción de Uniones Cruzadas

Se genera un producto cartesiano de las tablas EMPLOYEES y DEPARTMENTS utilizando la cláusula CROSS JOIN.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments;
```

4.4. Intaxis Oracle ANSI 92

Hasta la versión 8 la sintaxis permitida en Oracle respondía a es estándar ANSI 92 sin que se pudiesen utilizar las cláusulas vistas anteriormente.

Esta sintaxis es utilizada hasta el día de hoy por muchos desarrolladores, si bien la misma funciona tiene algunas consideraciones que hacen que sea más conveniente a los efectos de performance utilizar las cláusulas que responden al estándar ANSI 99.

```
SELECT last_name , department_name
FROM        employees e , departments d
WHERE    e.department_id = d.department_id;
Para las unidades de no igualdad la sintaxis es la siguiente SELECT
last_name , d.department_id , department_name
FROM        employees e , departments d
WHERE    e.department_id(+)= d.department_id
```

El operador de union externa (+) se coloca del lado donde la información es insuficiente.

En el ejemplo traerá como resultado el nombre del empleado de la tabla EMPLOYEES y de la tabla DEPARTMENTS traerá todos los departamentos incluso aquellos que no tienen asignados empleados.

Si el operador de unión externa (+) estuviese colocado después de la tabla DEPARTMENTS traería todos los empleados de la tabla EMPLOYEES incluidos aquellos que todavía no tuviesen asignado departamento.

El operador de unión externa (+) se puede colocar en cualquier extremo de la condición WHERE, pero no en ambos

Ejercicios

1. Escriba una consulta para que el departamento de recursos humanos genere las direcciones de todos los departamentos. Utilice las tablas LOCATIONS y COUNTRIES. Muestre en la salida el identificador de ubicación, el domicilio de calle, la ciudad, el estado o la provincia y el país. Utilice una unión natural para generar los resultados.

2. El departamento de recursos humanos necesita un informe de todos los empleados. Escriba una consulta para mostrar el apellido, el número de departamento y el nombre de departamento de todos los empleados.

3. Cree un informe que muestre el apellido del empleado y el número de empleado junto al apellido y número de supervisor del supervisor del empleado. Etiquete las columnas como Employee, Emp#, Manager y Mgr#, respectivamente. Guarde la sentencia SQL en un archivo de texto denominado lab_03_01.sql.

4 Modifique lab_03_01.sql para que muestre todos los empleados, incluido King, que no tiene supervisor. Ordene los resultados por el número de empleado. Guarde la sentencia SQL en un archivo de texto denominado lab_03_02sql. Ejecute la consulta en lab_03_02.sql

5. Cree un informe para el departamento de recursos humanos que muestre los apellidos, los números de departamento y todos los empleados que trabajan en el mismo departamento que un empleado dado. Proporcione a cada columna una etiqueta adecuada. Guarde el archivo de comandos en un archivo denominado lab_03_02.sql.

5. Funciones de Agregadas y Súper agregadas ORACLE

5.1. Objetivos

Esta lección trata más exhaustivamente las funciones. Se centra en la obtención de información de resumen (como, por ejemplo, medias) para grupos de filas.

Analiza cómo agrupar las filas de una tabla en juegos más pequeños y cómo especificar criterios de búsqueda para grupos de filas.

A diferencia de las funciones de una sola fila, las funciones de grupo operan en juegos de filas para dar un resultado por grupo. Estos juegos pueden abarcar toda la tabla o la tabla dividida en grupos.

5.1.1. Tipos de Funciones de Grupo

Cada una de las funciones acepta un argumento. La tabla siguiente identifica las opciones que puede utilizar en la sintaxis

Función	Descripción
AVG([DISTINCT|ALL]n)	Valor medio de n; se ignoran los valores nulos
COUNT({*|[DISTINCT|ALL]expr})	Número de filas, en las que expr se evalúa como algo no nulo (cuenta todas las filas seleccionadas que utilizan *, incluidas las duplicadas y las que contienen valores nulos)
MAX([DISTINCT|ALL]expr)	Valor máximo de expr; se ignoran los valores nulos
MIN([DISTINCT|ALL]expr)	Valor mínimo de expr; se ignoran los valores nulos
STDDEV([DISTINCT|ALL]x)	Desviación estándar de n; se ignoran los valores nulos
SUM([DISTINCT|ALL]n)	Suma los valores de n; se ignoran los valores nulos
VARIANCE([DISTINCT|ALL]x)	Varianza de n; se ignoran los valores nulos

5.2. Funciones de Grupo: Sintaxis

```
SELECT      [column,], group_function(column),...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

DISTINCT hace que la función considere únicamente valores no duplicados; ALL hace que considere todos los valores, incluidos los duplicados. El valor por defecto es ALL y, por tanto, no es necesario especificarlo.

Los tipos de datos para las funciones con un argumento expr pueden ser CHAR,

VARCHAR2, NUMBER o DATE.

Todas las funciones de grupo ignoran los valores nulos. Para sustituir con un valor los valores nulos, utilice las funciones NVL, NVL2 o COALESCE.

5.2.1. Uso de las Funciones AVG - SUM –MAX – MIN

Puede utilizar las funciones AVG, SUM, MIN y MAX en columnas que puedan almacenar datos numéricos. El ejemplo muestra la media, el más alto, el más bajo y la suma de los salarios mensuales de todos los representantes de ventas.

```
SELECT  AVG(salary), MAX(salary),MIN(salary), SUM(salary)
FROM        employees
WHERE       job_id LIKE '%REP%';
```

Las funciones AVG, SUM, VARIANCE y STDDEV se pueden utilizar sólo con tipos de datos numéricos.

Puede utilizar las funciones MAX y MIN para tipos de datos numéricos, de carácter y de fecha.

El ejemplo siguiente muestra el apellido del empleado primero y el del último de una lista alfabética de todos los empleados.

```
SELECT MIN(last_name), MAX(last_name)
```

```
FROM        emp loyees;
```

Las funciones AVG, SUM, VARIANCE y STDDEV se pueden utilizar sólo con tipos de datos numéricos. MAX y MIN no se pueden utilizar con tipos de datos LOB o LONG.

5.2.2. Función COUNT

La función COUNT tiene tres formatos:

- COUNT(*)
- COUNT(expr)
- COUNT(DISTINCT expr)

COUNT(*) devuelve el número de filas de una tabla que satisfacen el criterio de la sentencia SELECT, incluidas las filas duplicadas y las que contienen valores nulos de cualquiera de las columnas. Si se incluye una cláusula WHERE en la sentencia SELECT, COUNT(*) devuelve el número de filas que satisfacen la condición de la cláusula WHERE.

Por el contrario, COUNT(expr) devuelve el número de valores no nulos que hay en la columna identificada con expr.

COUNT(DISTINCT expr) devuelve el número de valores únicos no nulos que hay en la columna identificada con expr.

Ejemplo

```
SELECT      COUNT(*)
FROM        emp loyees
WHERE       depa r tmen t_ id = 50 ;
```

5.2.3. Palabra Clave DISTINCT

Utilice la palabra clave DISTINCT para suprimir el recuento de valores duplicados en una columna.

```
SELECT      COUNT(DISTINCT department_id)
FROM emp l oyees ;
```

5.2.4. Funciones de Grupo y Valores Nulos

Todas las funciones de grupo ignoran los valores nulos de la columna.

La función NVL fuerza a las funciones de grupo a incluir valores nulos.

Ejemplos

1. La media se calcula basándose sólo en las filas de la tabla en la que se almacena un valor válido en la columna COMMISSION_PCT. La media se calcula como comisión total que se paga a todos los empleados dividida por el número de empleados que reciben una comisión (cuatro).

```
SELECT AVG(commission_pct)
FROM   employees;
```

2. La media se calcula basándose en todas las filas de la tabla, independientemente de si se almacenan valores válidos en la columna COMMISSION_PCT. La media se calcula como comisión total que se paga a todos los empleados dividida por el número total de empleados de la compañía (20).

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

5.3. Creación de Grupos de Datos Cláusula GROUP BY

Hasta el momento, todas las funciones de grupo han tratado la tabla como un grupo de información de gran tamaño.

Hay ocasiones, sin embargo, en que debe dividir la tabla de información en grupos más pequeños. Esto se puede conseguir mediante la cláusula GROUP BY.

Puede utilizar la cláusula GROUP BY para dividir las filas de una tabla en grupos. Puede utilizar entonces las funciones de grupo para devolver información de resumen para cada grupo.

En la sintaxis:

group_by_expression especifica columnas cuyos valores determinan la base para agrupar filas

Instrucciones

Si incluye una función de grupo en una cláusula SELECT, no puede seleccionar además resultados individuales, a menos que la columna individual aparezca en la cláusula GROUP BY. Recibirá un mensaje de error si no incluye la lista de columnas en la cláusula GROUP BY.

- Mediante una cláusula WHERE, puede excluir filas antes de dividirlas en grupos.
- Debe incluir las columnas en la cláusula GROUP BY.
- No puede utilizar un alias de columna en la cláusula GROUP BY

5.3.1. Uso de la Cláusula GROUP BY

Cuando utilice la cláusula GROUP BY, asegúrese de que todas las columnas de la lista SELECT que no sean funciones de grupo se incluyan en la cláusula GROUP BY.

El ejemplo muestra el número de departamento y el salario medio de cada departamento. La sentencia SELECT, que contiene una cláusula GROUP BY, se evalúa así:

La cláusula SELECT especifica las columnas que se recuperarán, de este modo:

Columna de números de departamento de la tabla EMPLOYEES

Media de todos los salarios del grupo que especificó en la cláusula GROUP BY

La cláusula FROM especifica las tablas a las que debe acceder la base de datos: la tabla EMPLOYEES.

La cláusula WHERE especifica las filas que se recuperarán. Como no hay cláusula WHERE, se recuperarán todas las filas por defecto.

La cláusula GROUP BY especifica cómo se deben agrupar las filas. Las filas se agrupan por número de departamento, por lo que la función AVG que se aplica a la columna de salarios calcula el salario medio de cada departamento.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id ;
```

5.4. Agrupación por Más de Una Columna

En ocasiones, necesita ver los resultados para grupos dentro de grupos. El ejemplo muestra un informe que detalla el salario total que se paga a cada cargo en cada departamento.

La tabla EMPLOYEES se agrupa primero por número de departamento y después por cargo dentro de esa agrupación. Por ejemplo, los cuatro administrativos del departamento 50 se agrupan juntos y se crea un único resultado (salario total) para todos los administrativos del grupo.

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM   employees
GROUP BY department_id, job_id;
```

5.4.1. Consultas Ilegales que Utilizan Funciones de Grupo

Siempre que utilice una mezcla de elementos individuales (DEPARTMENT_ID) y funciones de grupo (COUNT) en la misma sentencia SELECT, debe incluir una cláusula GROUP BY que especifique los elementos individuales (en este caso, DEPARTMENT_ID).

Si falta la cláusula GROUP BY, aparece el mensaje de error "not a single-group group function" y un asterisco

(*) indica la columna con el error. Puede corregir el error agregando la cláusula GROUP BY

5.4.2. Restricción de Resultados de Grupos con la Cláusula HAVING

De la misma forma que utiliza la cláusula WHERE para restringir las filas que se seleccionarán, utilice la cláusula HAVING para restringir grupos. Para buscar el salario máximo de cada departamento que tenga un salario máximo superior a 10.000 pesos, necesita:

1. Buscar el salario medio de cada departamento agrupando por número de departamento.

2. Restringir los grupos a los departamentos con un salario máximo superior a 10.000 pesos.

Utilice la cláusula HAVING para especificar los grupos que se deben mostrar, con los que se restringen más los grupos basándose en la información de agregación.

En la sintaxis, group_condition restringe los grupos de filas devueltas a los grupos para los que es verdadera la condición especificada.

Oracle Server sigue estos pasos si se utiliza la cláusula HAVING:

1. Se agrupan las filas.

2. Se aplica al grupo la función de grupo.

3. Se muestran los grupos que satisfacen los criterios de la cláusula HAVING.

La cláusula HAVING puede ir delante de la cláusula GROUP BY, pero se recomienda poner primero la cláusula GROUP BY, porque resulta más lógico. Se forman los grupos y se calculan las funciones de grupo antes de que se aplique la cláusula HAVING a los grupos de la lista SELECT.

Uso de la Cláusula HAVING

El ejemplo de la diapositiva muestra números de departamento y salarios máximos de los departamentos con un salario máximo superior a 10.000 pesos.

Puede utilizar la cláusula GROUP BY sin utilizar la función de grupo de la lista SELECT. Si restringe filas basándose en el resultado de una función de grupo, debe tener una cláusula GROUP BY además de la cláusula HAVING.

El ejemplo siguiente muestra los números de departamento y salarios medios de los departamentos con un salario máximo superior a 10.000 pesos.:

```
SELECT department_id, AVG(salary)
FROM   employees
GROUP BY department_id
HAVING max(salary)>10000;
```

5.4.3. Anidamiento de Funciones de Grupo

Las funciones de grupo se pueden anidar hasta una profundidad de dos.

Por ejemplo

MAX(AVG(salary))

5.5. GROUP BY con los Operadores ROLLUP y CUBE

Especifique los operadores ROLLUP y CUBE en la cláusula GROUP BY de una consulta. El agrupamiento con ROLLUP genera un juego de resultados que contiene las filas agrupadas normales y las filas subtotales. La operación CUBE de la cláusula GROUP BY agrupa las filas seleccionadas basándose en los valores de todas las combinaciones posibles de expresiones de las especificaciones y devuelve una sola fila de información resumida para cada grupo. Puede utilizar el operador CUBE para generar filas de valores derivados de varias tablas.

Nota: Al trabajar con ROLLUP y CUBE, asegúrese de que las columnas que vayan después de GROUP BY tengan relaciones significativas y reales entre sí, ya que, de lo contrario, los operadores devolverán información irrelevante.

5.5.1. Operador ROLLUP

El operador ROLLUP proporciona agregados y superagregados para expresiones dentro de una sentencia GROUP BY. Los escritores de informes pueden utilizar el operador ROLLUP para extraer estadísticas e información de resumen de los juegos de resultados. Los agregados acumulativos se pueden utilizar en informes, diagramas y gráficos.

El operador ROLLUP crea agrupamientos moviéndose en una dirección, de derecha a izquierda, a lo largo de la lista de columnas especificada en la cláusula GROUP BY.

A continuación, aplica la función agregada a estos agrupamientos.

Nota: Para generar subtotales en n dimensiones (es decir, n columnas de la cláusula GROUP BY) sin un operador ROLLUP, se deben enlazar n+1 sentencias SELECT con UNION ALL. Esto hace que la ejecución de la consulta resulte ineficiente, ya que cada sentencia SELECT provoca acceso a tablas.

El operador ROLLUP recopila sus resultados con un solo acceso a tablas.

El operador ROLLUP es útil cuando hay muchas columnas implicadas en la generación de subtotales.

Los subtotales y los totales se generan con ROLLUP. CUBE genera totales también, pero acumula eficazmente en cada dirección posible, lo que genera datos de valores derivados de varias tablas.

Sintaxis

```
SELECT      [column,] group_function(column). . .
FROM        table
[WHERE      condition]
[GROUP BY[ROLLUP]      group_by_expression]
[HAVING     having_expression];
[ ORDER BY column ];
```

5.5.2. Ejemplo de un Operador ROLLUP

```
SELECT        department_id, job_id, SUM(salary)
FROM          employees
WHERE         department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

- Los salarios totales de todos los identificadores de puesto de un departamento para los departamentos cuyo identificador es menor que 60 se muestran mediante la cláusula GROUP BY.

El operador ROLLUP muestra:

- Salario total de cada departamento cuyo identificador es menor que 60
- Salario total de todos los departamentos cuyo identificador es menor que 60, independientemente de los identificadores de puesto

En este ejemplo,

1 indica un grupo totalizado tanto por DEPARTMENT_ID como por JOB_ID,

2 indica un grupo totalizado sólo por DEPARTMENT_ID y 3 indica la suma total.

El operador ROLLUP crea subtotales que acumulan desde el nivel más detallado hasta la suma total, después de la lista de agrupamiento especificada en la cláusula GROUP BY.

Primero, calcula los valores agregados estándar para los grupos especificados en la cláusula GROUP BY (en el ejemplo, la suma de salarios agrupados en cada puesto de un departamento).

A continuación, va creando subtotales de mayor nivel progresivamente, de derecha a izquierda en la lista de columnas de agrupamiento. (En el ejemplo, se calcula la suma de salarios para cada departamento, seguida de la suma de los salarios para todos los departamentos).

Dadas n expresiones en el operador ROLLUP de la cláusula GROUP BY, la operación da como resultado n + 1 (en este caso, 2 + 1 = 3) agrupamientos.

Las filas basadas en los valores de las n primeras expresiones se denominan filas o filas normales y las demás, filas superagregadas.

5.6. Operador CUBE

El operador CUBE es un conmutador adicional de la cláusula GROUP BY de una sentencia

SELECT. El operador CUBE se puede aplicar a todas las funciones agregadas, incluidas AVG, SUM, MAX, MIN y COUNT. Se utiliza para generar juegos de resultados que se suelen utilizar para informes de datos derivados de varias tablas. Mientras que ROLLUP genera sólo una fracción de posibles combinaciones de subtotales, CUBE genera subtotales para todas las posibles combinaciones de agrupamientos especificados en la cláusula GROUP BY y una suma total.

El operador CUBE se utiliza con una función agregada para generar filas adicionales en un juego de resultados. Las columnas incluidas en la cláusula GROUP BY son de referencia cruzada y se utilizan para generar un superjuego de grupos. La función agregada especificada en la lista de selecciones se aplica a estos grupos para generar valores de resumen para las filas superagregadas adicionales.

El número de grupos adicionales del juego de resultados lo determina el número de columnas incluidas en la cláusula GROUP BY.

De hecho, todas las posibles combinaciones de las columnas o las expresiones de la cláusula GROUP BY se utilizan para generar superagregados. Si tiene n columnas o expresiones en la cláusula GROUP BY, habrá 2n posibles combinaciones superagregadas.

Matemáticamente, estas combinaciones forman un cubo de n dimensiones, de ahí el nombre del operador.

Mediante la aplicación o herramientas de programación, estos valores superagregados se pueden proporcionar a diagramas y gráficos que expresarán los resultados y las relaciones eficazmente y de forma visual.

5.6.1. Ejemplo de un Operador CUBE

```
SELECT      department_id, job_id, SUM(salary)
FROM        employees
WHERE       department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

La salida de la sentencia SELECT del ejemplo se puede interpretar así:

- El salario total de todos los puestos dentro de un departamento (para los departamentos cuyo identificador es menor que 60) se muestra mediante la cláusula GROUP BY.
- Salario total de los departamentos cuyo identificador es menor que 60.
- Salario total de todos los puestos, independientemente del departamento.
- Salario total de los departamentos cuyo identificador es menor que 60, independientemente de los cargos.

En este ejemplo,

1 indica la suma total.

2 indica las filas totalizadas sólo por JOB_ID.

3 indica algunas de las filas totalizadas por DEPARTMENT_ID y JOB_ID. 4 indica algunas de las filas totalizadas sólo por DEPARTMENT_ID.

El operador CUBE también ha realizado la operación ROLLUP para mostrar los subtotales de los departamentos cuyo identificador es menor que 60 y el salario total de los de los departamentos cuyo identificador es menor que 60, independientemente de los cargos.

Además, el operador CUBE muestra el salario total de todos los puestos, independientemente del departamento.

Nota: De forma parecida al operador ROLLUP, para generar subtotales en n dimensiones (es decir, n columnas de la cláusula GROUP BY) sin un operador CUBE, se deben enlazar 2n sentencias SELECT con UNION ALL. Así pues, un informe de tres dimensiones requiere que se enlacen 23 = 8 sentencias SELECT con UNION ALL.

5.7. Función GROUPING

La función GROUPING se puede utilizar con los operadores CUBE o ROLLUP para entender mejor el modo en que se ha obtenido un valor de resumen.

La función GROUPING utiliza una sola columna como argumento. El valor de expr en la función GROUPING se debe corresponder con una de las expresiones de la cláusula GROUP BY. La función devuelve un valor de 0 ó 1.

Los valores devueltos por la función GROUPING son útiles para:

- Determinar el nivel de agregación de un subtotal dado; es decir, el grupo o los grupos en los que se basa el subtotal

Identificar si un valor NULL en la columna de expresiones de una fila del juego de resultado indica:

- Un valor NULL de la tabla base (valor NULL almacenado)

- Un valor NULL creado por ROLLUP o CUBE (como resultado de una función de grupo en esa expresión)

Un valor de 0 devuelto por la función GROUPING basándose en una expresión indica una de estas posibilidades:

Se ha utilizado la expresión para calcular el valor agregado.

El valor NULL de la columna de expresiones es un valor NULL almacenado.

Un valor de 1 devuelto por la función GROUPING basándose en una expresión indica una de estas posibilidades:

- No se ha utilizado la expresión para calcular el valor agregado.

- El valor NULL de la columna de expresiones se crea mediante ROLLUP o CUBE como resultado del agrupamiento.

5.7.1. Ejemplo de una Función GROUPING

```
SELECT depa r tmen t _ i d DEPT ID , j ob_ i d JOB ,
  SUM (sa l a ry ) ,
        GROUPING(depar tment_id)GRP_DEPT,
        GROUP ING(j ob_ i d) GRP_JOB
FROM emp l oyees
WHERE depa r tmen t_ id < 50
GROUP BY ROLLUP(department_id, job_id);
```

En el ejemplo de la diapositiva, observe el valor de resumen 4400 de la primera fila

Este valor de resumen es el salario total del identificador de puesto AD_ASST dentro del departamento 10. Para calcular este valor de resumen, se han tenido en cuenta las columnas DEPARTMENT_ID y JOB_ID. Así pues, se devuelve un valor de 0 para las expresiones GROUPING(department_id) y ROUPING(job_id).

Observe el valor de resumen 4400 de la segunda fila. Este valor es el salario total del departamento 10 y se ha calculado teniendo en cuenta la columna DEPARTMENT_ID; así pues, GROUPING(department_id) ha devuelto un valor de 0.

Como la columna JOB_ID no se ha tenido en cuenta para calcular este valor, se ha devuelto un valor de 1 para GROUPING(job_id). En la quinta fila, puede observar una salida parecida.

En la última fila, observe el valor de resumen 54800. Es el salario total para los departamentos cuyo identificador es menor que 50 y todos los cargos. Para calcular este valor de resumen, no se ha tenido en cuenta ninguna de las columnas DEPARTMENT_ID y JOB_ID. Así pues, se devuelve un valor de 1 para las expresiones GROUPING(department_id) y GROUPING(job_id).

5.7.2. GROUPING SETS

GROUPING SETS es una extensión adicional de la cláusula GROUP BY que se puede utilizar para especificar varios agrupamientos de datos. Esto facilita una agregación eficiente y, por tanto, facilita el análisis de datos en varias dimensiones.

Ahora se puede escribir una sola sentencia SELECT mediante GROUPING SETS para especificar varios agrupamientos (que también pueden incluir operadores ROLLUP o CUBE), en lugar de varias sentencias SELECT combinadas mediante los operadores UNION ALL.

Por ejemplo:

```
SELECT department_id, job_id, manager_id, AVG(salary) FROM
employees
GROUP BY
GROUPING SETS ((department_id, job_id, manager_id),
(department_id, manager_id), (job_id, manager_id));
Esta sentencia calcula agregados en tres agrupamientos:
(department_id, job_id, manager_id), (department_id,
manager_id) y
(job_id, manager_id)
```

Sin esta función, se requieren varias consultas combinadas junto con UNION ALL para obtener la salida de la sentencia SELECT anterior. Un enfoque multiconsulta resulta ineficiente, ya que requiere varias exploraciones de los mismos datos.

Compare el ejemplo anterior con la siguiente alternativa:

```
SELECT department_id, job_id, manager_id, AVG (salary)
FROM employees
GROUP BY CUBE(department_id, job_id, manager_id);
```

Esta sentencia calcula los 8 (2 *2 *2) agrupamientos, aunque sólo son los grupos (department_id, job_id, manager_id), (department_id, manager_id) y (job_id, manager_id) los que le interesan.

Otra alternativa es la siguiente sentencia:

```
SELECT department_id, job_id, manager_id, AVG(salary) FROM
employees
GROUP BY department_id, job_id, manager_id
UNION ALL
```

```
SELECT department_id, NULL, manager_id, AVG(salary) FROM
employees
GROUP BY department_id, manager_id
UNION ALL
SELECT NULL, job_id, manager_id, AVG(salary) FROM employees
GROUP BY job_id, manager_id;
```

Esta sentencia requiere tres exploraciones de la tabla base, lo que la hace ineficiente.

CUBE y ROLLUP se pueden considerar juegos de agrupamientos con semántica muy específica

CUBE(a, b, c) es equivalente a	GROUPING SETS ((a, b, c), (a, b), (a, c), (b, c),(a), (b), (c), ())
ROLLUP(a, b, c) es equivalente a	GROUPING SETS ((a, b, c), (a, b),(a), ())

5.7.3. GROUPING SETS: Ejemplo

```
SELECT department_id, job_id, manager_id, avg(salary)
  FROM employees
  GROUP BY GROUPING SETS ((department_id,job_id),
  (job_id,manager_id));
```

La consulta deejemplo calcula agregados en los dos agrupamientos.

La tabla se divide en los siguientes grupos:

- Identificadores de puesto, identificadores de supervisor
- Identificadores de departamento, identificadores de puesto

Se calculan los salarios medios de cada uno de estos grupos. El juego de resultados muestra el salario medio de cada uno de los dos grupos.

En la salida, el grupo se puede interpretar como:

- El salario medio de todos los empleados con el identificador de puesto AD_VP a las órdenes del supervisor 100 es de 17000.

- El salario medio de todos los empleados con el identificador de puesto AD_MGR a las órdenes del supervisor 101 es de 12000 y así sucesivamente.

El grupo siguiente en la salida se interpreta como:

El salario medio de todos los empleados con el identificador de puesto FI_MGR del departamento 100 es de 12000.

El salario medio de todos los empleados con el identificador de puesto FI_ACCOUNT en el departamento 100 es de 7920 y así sucesivamente.

5.7.4. Columnas Compuestas

Una columna compuesta es una recopilación de columnas que se tratan como una unidad durante el cálculo de agrupamientos. Especifique las columnas entre paréntesis como en la siguiente sentencia:

ROLLUP (a, (b, c), d)

Aquí, (b, c) forma una columna compuesta y se trata como una unidad.

Por lo general, las columnas compuestas son útiles en ROLLUP, CUBE y GROUPING SETS. Por ejemplo, en CUBE o ROLLUP, las columnas compuestas provocarían el salto de la agregación en determinados niveles.

Es decir,

GROUP BY ROLLUP(a, (b, c))es equivalente a

```
GROUP BY a , b , c
UN I ON ALL
GROUP BY a
UN I ON ALL
GROUP BY ( )
```

Aquí, (b, c) se trata como una unidad y ROLLUP no se aplica en (b, c).

Es como si se tiene un alias, por ejemplo, z, para (b, c) y la expresión GROUP BY se reduce a

GROUP BY ROLLUP(a, z).

Nota: GROUP BY( ) normalmente es una sentencia SELECT con valores NULL para las columnas a y b y sólo la función agregada. Esto se utiliza generalmente para generar sumas totales.

```
SELECT        NULL, NULL, aggregate_col
FROM          <table_name>
GROUP BY ( ) ;
```

5.7.5. Columnas Compuestas: Ejemplo

```
SELECT      department_id, job_id, manager_id,
              SUM(salary)
FROM        employees
GROUP BY ROLLUP ( department_id , ( job_id , manager_id ))
```

Esta consulta hace que Oracle Server calcule los siguientes agrupamientos:

1. (job_id, manager_id)
2. (department_id, job_id, manager_id)
3. (department_id)
4. Suma total

Si sólo le interesan grupos específicos, no puede limitar el cálculo a esos agrupamientos sin utilizar columnas compuestas. Con las columnas compuestas, esto es posible si se trata las columnas JOB_ID y MANAGER_ID como una unidad durante la acumulación. Las columnas entre paréntesis se tratan como una unidad durante los cálculos ROLLUP y CUBE. Esto se ilustra en el ejemplo de la diapositiva. Al poner las columnas JOB_ID y MANAGER_ID entre paréntesis, le indica a Oracle Server que trate JOB_ID y MANAGER_ID como una unidad, que es una columna compuesta.

5.8. Agrupamientos Concatenados

Los agrupamientos concatenados ofrecen una forma concisa de generar combinaciones de agrupamientos útiles.

Para especificar los agrupamientos concatenados, se muestran varios juegos de agrupamientos, cubos y acumulaciones, y se separan con comas.

A continuación se ofrece un ejemplo de juegos de agrupamientos concatenados:

GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)

Este ejemplo SQL define los siguientes agrupamientos:

(a, c), (a, d), (b, c), (b, d)

La concatenación de juegos de agrupamientos es muy útil por estos motivos:

Facilidad de desarrollo de consultas: No es necesario enumerar manualmente todos los agrupamientos.

Uso por las aplicaciones: El SQL generado por aplicaciones OLAP suele implicar la concatenación de juegos de agrupamientos, en la que cada juego de agrupamientos define los agrupamientos necesarios para una dimensión.

5.8.1. Agrupamientos Concatenados: Ejemplo

```
SELECT department_id, job_id, manager_id,
       SUM(salary)
FROM   employees
GROUP BY department_id,
ROLLUP(job_id),
CUBE(manager_id);
```

El ejemplo da como resultado los siguientes agrupamientos:

- (job_id, manager_id)
- (department_id, job_id, manager_id)

- (job_id)
- (department_id,manager_id)
- (department_id)

Práctica:

1 Busque el salario más alto, el más bajo, la suma y el salario medio de todos los empleados.

Etiquete las columnas como Maximum, Minimum, Sum y Average, respectivamente. Redondee los resultados al siguiente número entero. Guarde la sentencia SQL en un archivo de texto denominado lab_04_01.sql.

2 Modifique la consulta en lab_04_01.sql para mostrar el salario mínimo, el máximo, la suma y el salario medio para cada tipo de trabajo.

Vuelva a guardar como lab_04_02.sql. Ejecute la sentencia.

3 Cree una consulta que muestre el número total de empleados y, de dicho total, el número de empleados contratados en 1995, 1996, 1997 y 1998. Cree las cabeceras de columna adecuadas

4 .Escriba una consulta para mostrar lo siguiente para los empleados cuyo identificador de supervisor sea menor que 120:

Identificador de supervisor

Identificador de puesto y salario total de todos los identificadores de puesto de los empleados que estén a las órdenes del mismo supervisor

Salario total de esos supervisores

Salario total de esos supervisores, independientemente de los identificadores de puesto

5 .Escriba una consulta para mostrar lo siguiente para los empleados cuyo identificador de supervisor sea menor que 120:

Identificador de supervisor

Puesto y salarios totales de todos los puestos de empleados que estén a las órdenes del mismo supervisor

Salario total de esos supervisores

Valores desde varias tablas para mostrar el salario total de todos los puestos, independientemente del supervisor

Salario total independientemente de todos los cargos

6. Subconsultas

6.1. Objetivos

Al finalizar esta lección, debería estar capacitado para:

Definir sub-consultas

Describir los tipos de problemas que pueden resolver las subconsultas

Enumerar los tipos de subconsultas

Escribir subconsultas de una sola fila y de varias filas

En esta lección, obtendrá información sobre funciones más avanzadas de la sentencia SELECT. Puede escribir subconsultas en la cláusula WHERE de otra sentencia SQL para obtener valores basados en un valor condicional desconocido. Esta lección trata subconsultas de una sola fila y subconsultas de varias filas.

6.1.1. Uso de Subconsultas para Resolver Problemas

Supongamos que desea escribir una consulta para averiguar quién gana un salario mayor que el de un empleado determinado.

Para resolver este problema, necesita dos consultas: una para buscar cuánto gana ese emplead y una segunda para buscar quién gana más que esa cantidad.

Puede resolver este problema combinando las dos consultas, colocando una consulta dentro de la otra consulta.

La consulta interna (o subconsulta) devuelve un valor que se utiliza en la otra consulta (o consulta principal).

Utilizar una subconsulta es equivalente a realizar dos consultas secuenciales y utilizar el resultado de la primera consulta como valor de búsqueda en la segunda consulta.

6.1.2. Sintaxis de Subconsultas

Una subconsulta es una sentencia SELECT que está embebida en una cláusula de otra sentencia SELECT. Puede crear sentencias potentes sentencias a partir de otras más simples mediante subconsultas. Pueden resultar muy útiles si necesita seleccionar filas de una tabla con una condición que depende de los datos de la propia tabla.

Puede colocar la subconsulta en diferentes cláusulas SQL, como éstas:

- Cláusula WHERE
- Cláusula HAVING
- Cláusula FROM

En la sintaxis:

operador incluye una condición de comparación como, por ejemplo, >, = o IN

Nota: Hay dos clases de condiciones de comparación: operadores de una sola fila (>, =, >=, <, <>, <=) y operadores de varias filas (IN, ANY, ALL).

La subconsulta se suele conocer como sentencia SELECT anidada, subSELECT o SELECT interna. La subconsulta se suele ejecutar en primer lugar y se utiliza su salida para completar la condición de consulta para la consulta principal (o externa).

```
SELECT    columna|expresion
FROM      tabla
WHERE     columna operador
          (SELECT    columna|expresion
           FROM tabla);
```

6.1.3. Uso de Subconsultas

En el ejemplo, la consulta interna determina el salario del empleado Abel. La consulta externa toma el resultado de la consulta interna y lo utiliza para mostrar todos los empleados que ganan más que esta cantidad.

```
SELECT last_name
FROM      employees
```

```
WHERE      salary>
(SELECT salary
  FROM         employees
  WHERE      last_name = 'Abel');
```

6.1.4. Instrucciones para el Uso de Subconsultas

Las subconsultas deben ir entre paréntesis.

Sitúe las subconsultas a la derecha de la condición de comparación para que se lean mejor.

Con Oracle8i y versiones posteriores, se puede utilizar una cláusula ORDER BY y es necesaria en la subconsulta para realizar análisis de los N principales.

En Oracle8i, las subconsultas no podían contener una cláusula ORDER BY. Sólo se podía utiliza una cláusula ORDER BY para una sentencia SELECT; si se especificaba, tenía que ser la última cláusula de la sentencia SELECT principal.

Se utilizan dos clases de condiciones de comparación en las subconsultas: operadores de una sola fila y operadores de varias filas.

6.1.5. Tipos de Subconsultas

Subconsultas de una sola fila:

- Consultas que devuelven sólo una fila de la sentencia SELECT interna.
- Subconsultas de varias filas:

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

<>	Distinto de

Consultas que devuelven más de una fila de la sentencia SELECT interna.

- Nota: También hay subconsultas de varias columnas, que son consultas que devuelven más de una columna de la sentencia SELECT interna.

6.1.6. Subconsultas de Una Sola Fila

Una subconsulta de una sola fila es la que devuelve una fila de la sentencia SELECT interna. Este tipo de subconsulta utiliza un operador de una sola fila. La diapositiva ofrece una lista de operadores de una sola fila.

Ejemplo

Muestre los empleados cuyos identificadores de puesto sean los mismos que los del empleado 141:

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
(SELECT job_id
FROM   employees
WHERE  employee_id = 141);
```

Operadores de comparación de una sola fila

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	Distinto de

6.1.7. Ejecución de Subconsultas de Una Sola Fila

Una sentencia SELECT se puede considerar un bloque de consulta. El muestra los empleados cuyos identificadores de puesto son iguales que el del empleado 141 y cuyos salarios son mayores que el del empleado 143.

El ejemplo consta de tres bloques de consulta: la consulta externa y dos consultas internas. Los bloques de consultas internas se ejecutan en primer lugar, con lo que se crean los resultados de consulta ST_CLERK y 2500, respectivamente. A continuación se procesa el bloque de la consulta externa, utilizando los valores que devolvieron las consultas internas para completar sus condiciones de búsqueda.

Las dos consultas internas devuelven valores únicos (ST_CLERK y 2500, respectivamente), por lo cual esta sentencia SQL se denomina subconsulta de una sola fila.

Nota: Las consultas externas e internas pueden obtener datos de tablas diferentes.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id = ( SELECT job_id
FROM   employees
WHERE  employee_id = 141 )
AND salary > ( SELECT salary
FROM   employees
WHERE  employee_id = 143 );
```

6.1.8. Uso de Funciones de Grupo en una Subconsulta

Puede mostrar datos de una consulta principal mediante una función de grupo en una subconsulta para devolver una sola fila. La subconsulta está entre paréntesis y se coloca después de la condición de comparación.

El ejemplo muestra el apellido de empleado, el identificador y el salario de todos los empleados cuyo salario sea igual al salario mínimo. La función de grupo MIN devuelve un solo valor (2500) a la consulta externa.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary =(SELECT MIN(salary)
FROM   employees) ;
```

6.1.9. La Cláusula HAVING con Subconsultas

Puede utilizar subconsultas no sólo en la cláusula WHERE, sino también en la cláusula

HAVING. Oracle Server ejecuta la subconsulta y los resultados se devuelven en la cláusula HAVING de la cláusula principal.

La sentencia SQL del ejemplo muestra todos los departamentos que tienen un salario mínimo mayor que el del departamento 50.

```
SELECT department_id, MIN(salary)
FROM        employees
GROUP BY department_id
HAVING MIN(salary) > (SELECT MIN(salary)
FROM employees
WHERE department_id = 50 ) ;
```

6.2. Errores clásicos en subconsultas

Un error común en las subconsultas se produce cuando se devuelve más de una fila para una subconsulta de una sola fila.

En la sentencia SQL del ejemplo, la subconsulta contiene una cláusula GROUP BY, lo que implica que devolverá varias filas, una para cada grupo que encuentre. En este caso, los resultados de la subconsulta son 4400, 5000, 2500, 4200, 7000, 17000 y 8300.

La consulta externa toma estos resultados y los utiliza en su cláusula WHERE. La cláusula WHERE contiene un operador de igualdad (=), operador de comparación de una sola fila que espera sólo un valor. El operador = no puede aceptar más de un valor de la subconsulta y, por tanto, genera el error.

```
SELECT employee_id, last_name
FROM        employees
WHERE      salary = (SELECT MIN(salary)
FROM        employees
GROUP BY department_id);
ERROR at line 4:
ORA-01427: single-row subquery returns more than one row
```

Para corregir este error, cambie el operador = a IN.

Problemas con las Subconsultas

Un problema habitual de las subconsultas se produce cuando la consulta interna no devuelve ninguna fila.

En la sentencia SQL de la diapositiva, la subconsulta contiene una cláusula WHERE.

Presumiblemente, la intención es buscar el empleado cuyo apellido es Haas sentencia es correcta, pero no selecciona ninguna fila cuando se ejecuta.

No hay ningún empleado llamado Haas. Así pues, la subconsulta no devuelve ninguna fila. La consulta externa toma los resultados de la subconsulta (nulos) y utiliza esos resultados en su cláusula WHERE. La salida externa no encuentra ningún empleado cuyo identificador de puesto sea igual a nulo, así que no devuelve ninguna fila. Si existiera un puesto con un valor nulo, la fila no se devolvería porque la comparación de dos valores nulos genera un valor nulo; así pues, la condición WHERE no es verdadera.

```
SELECT l as t _name , job_ id
FROM          emp loyees
HERE        j ob_ i d = (SELECT j ob_ id
FROM emp l oyees
WHERE las t_name = 'Haas ' ) ;
no rows se l ec ted
```

6.2.1. Subconsultas de Varias Filas

Las subconsultas que devuelven más de una fila se denominan subconsultas de varias filas. Puede utilizar operadores de varias filas, en lugar de los de una sola fila, con las subconsultas de varias filas El operador de varias filas espera uno o más valores:

```
SELECT last_name, salary, departmen t_id
FROM emp l oyees
WHERE salary IN (SELECT MIN(sa lary)
FROM emp l oyees
GROUP BY depa r tmen t _ id ) ;
```

Ejemplo

Busque los empleados que ganan un salario igual al salario mínimo para cada departamento.

La consulta interna se ejecuta en primer lugar, generando un resultado de consulta.

A continuación se procesa el bloque de consulta principal y utiliza los valores que devolvió la consulta interna para completar su condición de búsqueda. De hecho, la consulta principal aparece en Oracle Server de este modo:

```
SELECT last_name, salary, department_id
FROM   employees
WHERE  salary IN (2500, 4200, 4400 , 5000 , 7000 , 8300 ,8500 ,
17000 ) ;
```

6.2.2. Operadores de varias filas

Operador	Significado
IN	Igual que algún miembro de la lista
ANY	Compara el valor con cada valor devuelto
ALL	Compara el valor con todos los valores devueltos por la subconsulta

6.2.3. Uso del Operador ANY en Subconsultas de Varias Filas

El operador ANY compara un valor con cada valor devuelto por la subconsulta.

El ejemplo muestra empleados que no son programadores informáticos y cuyo salario es menor que el de cualquier programador informático.

El salario máximo que gana un programador es de 9.000 dólares.

Operador	Significado
ANY	es equivalente a IN
<ANY	Significa menos que el máximo
>ANY	significa más que el mínimo

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
               (SELECT sala ry
```

```
                    FROM        employees
                    WHERE       job_id = 'IT_PROG')
AND job_id <> 'IT_PROG'
```

6.2.4. Uso del Operador ALL en Subconsultas de Varias Filas

El operador ALL compara un valor con todos los valores devueltos por una subconsulta. El ejemplo de la diapositiva muestra los empleados cuyos salarios son menores que los de todos los empleados con el identificador de puesto IT_PROG y cuyos puestos no son IT_PROG.

El operador NOT se puede utilizar con los operadores IN, ANY y ALL.

Operador	Significado
ALL	es equivalente a
>ALL	significa más que el máximo
<ALL	significa menos que el mínimo

```
SELECT employee_id, last_name, job_id, salary
FROM        employees
WHERE       salary < ALL (SELECT salary
FROM        employees
WHERE       job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

6.2.5. Devolución de Valores Nulos en el Juego Resultante de una Subconsulta

La sentencia SQL del ejemplo intenta mostrar todos los empleados que no tienen subordinados. Por lógica, esta sentencia SQL debería haber devuelto 12 filas. Sin embargo, la sentencia SQL no devuelve ninguna fila. Uno de los valores devueltos por la consulta interna es un valor nulo y por eso la consulta entera no devuelve ninguna fila.

El motivo es que todas las condiciones que comparan un valor nulo dan como resultado un valor nulo. Así pues, siempre que sea probable que en el juego de resultados de una subconsulta haya valores nulos, no utilice el operador NOT IN.

El operador NOT IN es equivalente a <> ALL.

Observe que no hay problema en que en el juego de resultados de una subconsulta haya valores nulos si se utiliza el operador IN. El operador IN es equivalente a =ANY. Por ejemplo, para mostrar los empleados que tienen subordinados, utilice esta sentencia SQL:

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id  IN
(SELECT mgr.manager_id
FROM   employees mgr);
```

De forma alternativa, se puede incluir una cláusula WHERE en la subconsulta para mostrar todos los empleados que no tienen subordinados:

```
SELECT last_name FROM employees
WHERE  employee_id NOT IN
( SELECT manager_id
   FROM   employees
   WHERE  manager_id IS NOT NULL );
```

6.2.6. Subconsultas de Varias Columnas

Hasta ahora, ha escrito subconsultas de una sola fila y subconsultas de varias filas en las que sólo se devuelve una columna mediante la sentencia interna SELECT y esto se utiliza para evaluar la expresión de la sentencia SELECT principal. Si desea comparar dos o más columnas, debe escribir una cláusula WHERE compuesta mediante operadores lógicos.

Mediante subconsultas de varias columnas, puede combinar condiciones WHERE duplicadas en una sola cláusula WHERE.

Sintaxis

```
SELECT   columna, columna, ...
FROM tabla
WHERE (columna, columna, ...) IN
```

```
    (SELECT columna, columna, ...
FROM    tabla
WHERE   condicion);
```

El ejemplo muestra que los valores de MANAGER_ID y DEPARTMENT_ID de la consulta principal se están comparando con los valores MANAGER_ID y DEPARTMENT_ ID recuperados por la subconsulta. Como el número de columnas que se están comparando es superior a uno, el ejemplo se califica como subconsulta de varias columnas.

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   (manager_id, department_id) IN
(SELECT manager_id, department_id
  FROM employees
  WHERE employee_id IN (199 , 174 ) )
```

6.2.7. Comparaciones de Columnas

Las comparaciones de columnas en una subconsulta de varias columnas pueden ser entre pares y entre no pares.

En el ejemplo de la siguiente diapositiva, se ejecuta una comparación entre pares en la cláusula WHERE. Cada fila candidata de la sentencia SELECT debe tener las dos mismas columnas MANAGER_ID y DEPARTMENT_ID que los empleados con EMPLOYEE_ID 199 ó 174.

Una subconsulta de varias columnas también puede ser una comparación entre no pares. En una comparación entre no pares, cada columna de la cláusula WHERE de la sentencia SELECT principal se compara individualmente con varios valores recuperados por la sentencia SELECT interna. Las columnas individuales se pueden corresponder con cualquier valor recuperado por la sentencia SELECT interna. Pero colectivamente, se deben satisfacer todas las condiciones múltiples de la sentencia SELECT principal para que se muestre la fila. El ejemplo de la página siguiente ilustra una comparación entre pares.

```
SELECT  employee_id, manager_id, department_id
FROM    employees
WHERE   (manager_id, department_id) IN
(SELECT manager_id, department_id
  FROM employees
  WHERE employee_id IN (199 , 174 ) )
```

6.2.8. Subconsulta de Comparación entre Pares

El ejemplo muestra una subconsulta de varias columnas, ya que la subconsulta devuelve más de una columna.

Compara los valores de la columna MANAGER_ ID y de la columna DEPARTMENT_ID de cada fila en la tabla EMPLOYEES con los valores de la columna MANAGER_ID y la columna DEPARTMENT_ID para los empleados con EMPLOYEE_ID 199 ó 174.

Primero, se ejecuta la subconsulta para recuperar los valores de MANAGER_ID y DEPARTMENT_ID para los empleados con EMPLOYEE_ID 199 ó 174. Estos valores se comparan con la columna MANAGER_ID y la columna DEPARTMENT_ID de cada fila de la tabla EMPLOYEES. Si los valores se corresponden, se muestra la fila. En la salida, no se mostrarán los registros de los empleados con EMPLOYEE_ID 199 ó 174.

```
SELECT    employee_id, manager_id, department_id
FROM      employees
WHERE     (manager_id, department_id)  IN
(SELECT manager_id, department_id
  FROM employees
  WHERE employee_id IN (199 , 174 ) )
```

6.2.9. Subconsulta de Comparación entre No Pares

El ejemplo muestra una comparación entre no pares de las columnas.

Muestra los valores de EMPLOYEE_ID, MANAGER_ID y DEPARTMENT_ID de cualquier empleado cuyoidentificador de supervisor se corresponda con los identificadores de supervisor de los empleados cuyos identificadores sean 174 ó 199 y los valores de DEPARTMENT_ID se correspondan con los identificadores de departamento cuyo identificador de empleado sea 174 ó 199.

Primero, se ejecuta la subconsulta para recuperar los valores de MANAGER_ID para los empleados con el valor de EMPLOYEE_ID 199 ó 174. De forma parecida, se ejecuta la segunda subconsulta para recuperar los valores de DEPARTMENT_ID para los empleados con el valor de EMPLOYEE_ID 199 ó 174. Los valores recuperados de las columnas MANAGER_ID y DEPARTMENT_ID se comparan con las columnas MANAGER_ID y DEPARTMENT_ID de cada fila de la tabla EMPLOYEES. Si la columna MANAGER_ID de la fila de la tabla EMPLOYEES se corresponde con cualquiera de los valores de MANAGER_ID recuperados por la subconsulta interna y si la columna DEPARTMENT_ID

de la fila de la tabla EMPLOYEES se corresponde con cualquiera de los valores de DEPARTMENT_ID recuperados por la segunda subconsulta, se muestra el registro.

```
SELECT      employee_id, manager_id, department_id
FROM        employees
WHERE         manage r_ id IN
(SELECT      manager_id
  FROMemp l oyees
  WHERE emp loyee_ i d I N (174 ,199 ) )
AND department_i d  IN
(SELECT  depar tment_id
FROM        emp loyees
WHERE   empl oyee_id  IN  (174,199))
AND employee_i d NOT IN(174,199);
```

6.2.10. Subconsutla en una cláusula FROM

Puede utilizar una Subconsultas en la cláusula FROM de una sentencia SELECT, que es muy similar al modo en que se utilizan las vistas.

Una subconsulta en la cláusula FROM también se las denomina VOL (View On Line o Vista en línea).

La persistencia de este tipo de vistas solamente para esa consulta y se constituye en el origen de la misma.

```
SELECT a . las t _name , a .sa l a ry , a .depa r tmen t _ id , b .sa lavg
FROM emp l oyees a
JO I N (SELECT depa r tmen t_ i d , AVG (sa la ry ) sa l avg
FROM emp l oyees
GROUP BY depa r tmen t _ id ) b
ON a.department_id=b.depar tment_i d
WHERE a. sa l ary>b.sa lavg
```

6.2.11. Expresiones de Subconsultas Escalares

Una subconsulta que devuelve exactamente un valor de columna de una fila se conoce también como subconsulta escalar. Las subconsultas de varias columnas que se escriben

para comparar dos o más columnas, mediante una cláusula WHERE compuesta y operadores lógicos, no se califican como subconsultas escalares.

El valor de la expresión de subconsulta escalar es el valor del elemento de lista de selección de la subconsulta. Si la subconsulta devuelve 0 filas, el valor de la expresión de subconsulta escalar es NULL. Si la subconsulta devuelve más de una fila, Oracle Server devuelve un error.

Oracle Server siempre ha soportado el uso de una subconsulta escalar en una sentencia SELECT. Puede utilizar subconsultas escalares en:

La parte de expresión y condición de DECODE y CASE

Todas las cláusulas de SELECT excepto GROUP BY

La cláusula SET y la cláusula WHERE de una sentencia UPDATE

Sin embargo, las subconsultas escalares no son expresiones válidas en los siguientes casos:

Como valores por defecto para columnas y expresiones de comprobación aleatoria para agrupamientos

En la cláusula RETURNING de sentencias DML

Como base de un índice basado en funciones

En cláusulas GROUP BY, restricciones CHECK, condiciones WHEN

En cláusulas CONNECT BY

El primer ejemplo muestra que las subconsultas escalares se pueden utilizar en expresiones CASE. La consulta interna devuelve el valor 20, que es el identificador de departamento del departamento cuyo identificador de ubicación es 1800. La expresión CASE de la consulta externa utiliza el resultado de la consulta interna para mostrar el identificador de empleado, los apellidos y un valor de Canadá o EE.UU., dependiendo de si el identificador de departamento del registro recuperado por la consulta externa es 20 o no.

```
SELECT emp l oyee_ id , l as t_name ,
CASE
WHEN depa r t men t _ id = (
SELECT depa r t men t_ id
```

```
FROM depa r t men t s
WHERE loca t ion_ id = 1800 )
THEN 'Canada '
ELSE  'USA ' END) l oca t i on
FROM     emp loyees ;
```

El segundo ejemplo de la diapositiva demuestra que las subconsultas escalares se pueden utilizar en la cláusula ORDER BY. En el ejemplo, se ordena la salida basándose en DEPARTMENT_NAME al hacer corresponder el valor de DEPARTMENT_ID de la tabla EMPLOYEES con el valor de DEPARTMENT_ID de la tabla DEPARTMENTS. Esta comparación se realiza en una subconsulta escalar en la cláusula ORDER BY

```
SELECT emp l oyee_ id , l as t_name
FROM emp l oyees e
ORDER BY ( SELECT depa r tmen t_name
                FROM  depa r t men t s d
WHERE e .depa r tmen t_ i d = d .depa r t men t _ id ) ;
```

6.3. Subconsultas Correlacionadas

Oracle Server realiza una consulta correlacionada cuando la subconsulta hace referencia a una columna desde una tabla a la que se hace referencia en la sentencia principal. Una subconsulta correlacionada se evalúa una vez para cada fila procesada por la sentencia principal. La sentencia principal puede ser una sentencia SELECT, UPDATE o DELETE.

Subconsultas Anidadas frente a Subconsultas Correlacionadas

Con una subconsulta anidada normal, la consulta SELECT interna se ejecuta primero y una vez, y devuelve valores que serán utilizados por la consulta principal. Sin embargo, una subconsulta correlacionada se ejecuta una vez para cada fila candidata considerada por la consulta externa. Dicho de otro modo, la consulta interna está controlada por la consulta externa.

Ejecución de Subconsultas Anidadas

La consulta interna se ejecuta primero y encuentra un valor.

La consulta externa se ejecuta una vez y utiliza el valor de la consulta interna.

Ejecución de Subconsultas Correlacionadas

Obtenga una fila candidata (recuperada por la consulta externa).

Ejecute la consulta interna mediante el valor de la fila candidata.

Utilice los valores resultantes de la consulta interna para calificar o descalificar la fila candidata.

Repita el proceso hasta que no queden filas candidatas.

Una subconsulta correlacionada es una forma de leer todas las filas de una tabla y comparar los valores de cada fila con datos relacionados. Se utiliza siempre que una subconsulta deba devolver un resultado o un juego de resultados diferente para cada fila candidata considerada por la consulta principal. Dicho de otro modo, utilice una subconsulta correlacionada para responder a una pregunta de varias partes cuya respuesta dependa del valor de cada fila procesada por la sentencia principal.

Oracle Server realiza una consulta correlacionada cuando la subconsulta hace referencia a una columna desde una tabla de la consulta principal.

Nota: Puede utilizar los operadores ANY y ALL en una subconsulta correlacionada

6.3.1. Uso de Subconsultas Correlacionadas

El ejemplo se determina qué empleados ganan más que el salario medio de su departamento. En este caso, la subconsulta correlacionada calcula específicamente el salario medio de cada departamento.

Como la consulta externa y la interna utilizan la tabla EMPLOYEES en la cláusula FROM, se asigna un alias a EMPLOYEES en la sentencia SELECT externa por motivos de claridad. El alias no sólo hace que toda la sentencia SELECT resulte más legible, sino que sin él, la consulta no funcionaría correctamente, ya que la sentencia interna no podría distinguir la columna de la tabla interna de la columna de la tabla externa.

```
SELECT last_name, salary, department_id
FROM   employees outer
WHERE  salary >
(SELECT AVG(salary)
  FROM employees
WHERE  department_id = outer.department_id);
```

6.3.2. Uso del Operador EXISTS

Con las sentencias SELECT anidadas, son válidos todos los operadores lógicos. Además, puede utilizar el operador EXISTS. Este operador se utiliza frecuentemente con subconsultas correlacionadas para probar si existe un valor recuperado por la consulta externa en el juego de resultados de los valores recuperados por la consulta interna. Si la subconsulta devuelve al menos una fila, el operador devuelve TRUE. Si el valor no existe, devuelve FALSE. De forma análoga, NOT EXISTS prueba si un valor recuperado por la consulta externa no forma parte del juego de resultados de los valores recuperados por la consulta interna.

El operador EXISTS asegura que la búsqueda en la consulta interna no continúe cuando se encuentre al menos una correspondencia para el número de empleado y de supervisor

```
mediante la condición:
WHERE manager_id = outer.employee_id.
```

Observe que la consulta SELECT interna no necesita devolver un valor específico, por lo que se puede seleccionar una constante

```
SELECT employee_id, last_name, job_id, department_id
FROM employees outer
WHERE EXISTS ( SELECT 'X'
FROM          employees
WHERE     manager_id =outer.employee_id);
```

Uso del Operador NOT EXISTS

Se puede utilizar una construcción NOT IN como alternativa para un operador NOT EXISTS, como se muestra en el siguiente ejemplo:

```
SELECT  department_id, department_name
FROM     departments
WHERE   department_id NOT IN
     (SELECT department_id
     FROM     employees);
```

Sin embargo, NOT IN se evalúa como FALSE si cualquier miembro del juego es un valor NULL. Por tanto, la consulta no devolverá ninguna fila incluso aunque haya filas en la tabla DEPARTMENTS que satisfagan la condición WHERE.

6.3.3. Cláusula WITH

Mediante la cláusula WITH, puede definir un bloque de consulta antes de utilizarlo en una consulta. La cláusula WITH (conocida formalmente como subquery_factoring_clause) le permite reutilizar el mismo bloque de consulta en una sentencia SELECT cuando se produce más de una vez dentro de una consulta compleja. Esto resulta particularmente útil cuando una consulta tiene muchas referencias al mismo bloque de consulta y hay presentes uniones y agregaciones.

Mediante la cláusula WITH, puede reutilizar la misma consulta cuando resulta caro evaluar el bloque de consulta y se produce más de una vez dentro de una consulta compleja. Mediante la cláusula WITH, Oracle Server recupera los resultados de un bloque de consulta y los almacena en el tablespace temporal del usuario. Esto puede mejorar el rendimiento.

Ventajas de la Cláusula WITH

Hace que la consulta resulte fácil de leer

Evalúa una cláusula sólo una vez, incluso aunque aparezca varias veces en la consulta

En la mayoría de los casos puede mejorar el rendimiento para las consultas grandes

6.3.4. Cláusula WITH: Ejemplo

El problema de la diapositiva requeriría los siguientes cálculos intermedios:

1. Calcule el salario total para todos los departamentos y almacene el resultado mediante una cláusula WITH.

2. Calcule el salario medio de todos los departamentos y almacene el resultado mediante una cláusula WITH.

3. Compare el salario total calculado en el primer paso con el salario medio calculado en el Segundo paso. Si el salario total de un departamento en particular es mayor que el salario medio de todos los departamentos, muestre el nombre de departamento y el salario total de ese departamento.

Se utiliza sólo con sentencias SELECT.

Un nombre de consulta es visible para todos los bloques de consulta del elemento WITH (incluidos los bloques de subconsulta) definidos después de éste y el propio bloque de consulta (incluidos los bloques de subconsulta).

Cuando el nombre de consulta es igual que un nombre de tabla existente, el analizador busca de dentro a fuera y el nombre de bloque de consulta tiene prioridad sobre el nombre de tabla.

La cláusula WITH puede contener más de una consulta. Cada consulta se separa entonces con una coma.

```
WITH
dept_costs AS ( SELECT d.department_name, SUM(e.salary) AS
dept_total
                FROM employees e, departments d
WHERE e.department_id = d.department_id
GROUP BY d.department_name),
avg_cost AS ( SELECT SUM(dept_total)/COUNT(*) AS dept_avg
FROM dept_costs)
SELECT *
FROM dept_costs
WHERE dept_total >
( SELECT dept_avg
FROM avg_cost)
ORDER BY department_name;
```

Práctica

1. El departamento de recursos humanos necesita una consulta que pida al usuario el apellido de un empleado. La consulta muestra entonces el apellido y la fecha de contratación de cualquier empleado del mismo departamento que aquel cuyo apellido se suministre (excepto ese empleado).

2. Escriba una consulta que muestre el número de empleado y el apellido de todos los empleados que trabajen en un departamento en que haya algún empleado cuyo apellido contenga una u. Guarde la sentencia SQL en un archivo de texto denominado lab_05_01.sql. Ejecute la consulta.

3. Modifique la consulta en lab_05_01.sql para mostrar el número de empleado, el apellido y el salario de todos los empleados que ganen más que el salario medio y que trabajen en un departamento con algún empleado cuyo apellido contenga una u. Vuelva a guardar como lab_05_02.sql. Ejecute la nueva sentencia

4. Cree una consulta para mostrar el apellido, la fecha de contratación y el salario de todos los empleados que tengan el mismo salario y la misma comisión que Kochhar.

Nota: No muestre a Kochhar en el juego de resultados.

5 Busque todos los empleados que no sean supervisores. a. Hágalo primero mediante el operador NOT EXISTS.

6. Escriba una consulta para mostrar los nombres de departamento de los departamentos cuyo costo de salario total supere un octavo (1/8) del costo de salario total de toda la compañía. Utilice la cláusula WITH para escribir esta consulta. Asigne a la consulta el nombre SUMMARY

7. Manipulación de Datos

7.1. Objetivos

En esta lección aprenderá a insertar filas en un tabla, actualizar datos y suprimir filas existentes de una tabla. También como garantizar la consistencia de una transacción con cláusulas de control de datos como COMMIT, ROLLBACK y SAVEPOINT.

7.1.1. Lenguaje de Manipulación de Datos

El lenguaje de manipulación de datos (DML) es una pieza central de SQL. Cuando desee agregar, actualizar o suprimir datos de la base de datos, ejecute una sentencia DML. Una recopilación de sentencias DML que forma una unidad de trabajo lógica se llama transacción.

Considere la base de datos de un banco. Cuando un cliente del banco transfiere dinero de una cuenta de ahorros a una de cheques, la transacción puede constar de tres operaciones distintas: disminución de la cuenta de ahorros, aumento de la cuenta de cheques y registro de la transacción en el diario de transacciones. Oracle Server debe garantizar que las tres sentencias SQL se realizan para mantener las cuentas en el equilibrio correcto. Si algo evita la ejecución de una de las sentencias de la transacción, las otras sentencias se deben deshacer.

7.1.2. Sintaxis de la Sentencia INSERT

```
INSERT INTO tabla [(columna [, columna...])]
VALUES (valor [, valor...]);
```

Puede agregar filas nuevas a una tabla emitiendo la sentencia INSERT.

En la sintaxis:

table es el nombre de la tabla.

columna es el nombre de la columna de la tabla que se ha derellenar.

value es el valor correspondiente para la columna.

Nota: Esta sentencia con la cláusula VALUES sólo agrega una fila cada vez a una tabla.

7.1.3. Inserción de Nuevas Filas

Como puede insertar una fila nueva que contenga valores para cada columna, no se requiere la lista de columnas en la cláusula INSERT.

Sin embargo, si no utiliza la lista de columnas, los valores se deben listar de acuerdo con el orden por defecto de las columnas de la tabla y se debe proporcionar un valor para cada columna.

Para lograr una mayor claridad, utilice la lista de columnas en la cláusula INSERT.

Escriba los valores de caracteres y de fecha entre comillas simples; no se recomienda escribir los valores numéricos entre comillas simples.

Los valores numéricos no se deben escribir entre comillas simples, pues, si se hace, puede tener lugar una conversión implícita para los valores numéricos asignados a columnas del tipo de dato NUMBER.

```
INSERT INTO departments(department_id, department_name,
    manager_id, location_id)
VALUES          (60 , 'Pub l ic Re la t ions ' , 100 , 1600 ) ;
```

7.1.4. Inserción de Filas con Valores Nulos

Asegúrese de que puede utilizar valores nulos en la columna de destino verificando el estado de Null con el comando DESCRIBE.

Oracle Server fuerza automáticamente todos los tipos de dato, rangos de datos y restricciones de integridad de datos. Toda columna no listada explícitamente obtiene un valor nulo en la fila nueva.

Errores Habituales que se pueden producir durante la entrada del usuario:

- Falta un valor obligatorio para una columna NOT NULL.
- Un valor duplicado viola la restricción de unicidad.
- Se viola una restricción de clave ajena.
- Se viola una restricción CHECK.
- El tipo de dato no coincide.
- El valor es demasiado ancho para la columna.
-

Método	Descripción
• Implícito	Omita la columna de la lista de columnas.
• Explícito	Especifique la palabra clave NULL en la lista VALÚES, especifique la cadena vacía (' ') en la lista valúes para cadenas de caracteres y fechas.

•

```
INSERT INTO departments
VALÚES (100, 'Finance', NULL, NULL);
```

7.2. Inserción de Valores Especiales

Puede utilizar funciones para introducir valores especiales en la tabla.

En el ejemplo se registra información del empleado Popp en la tabla EMPLOYEES, se proporciona la fecha y la hora actuales en la columna HIRE_DATE y se utiliza la función SYSDATE para la fecha y la hora actuales.

También puede utilizar la función USER al insertar filas en una tabla.

Esta función registra el nombre de usuario actual.

```
INSERT INTO employees (employee_id,first_name,
last_name , job_id ,
hire_date,      salary,
commission_pct, manager_id,
department_id)
VALUES      (113,  'Louis', 'Popp' ,  'AC_ACCODNT',
  SYSDATE , 6900 ,NÜLL , 205 , 100) ;
```

Confirmación de Adiciones a la Tabla

```
SELECT employee_id, last_name,  job_id,
          hire_date,      commission_pct
FROM      employees
WHERE     employee_id =    113;
```

7.2.1. Inserción de Valores de Fecha Específicos

El formato DD-MON- YY se utiliza normalmente para insertar un valor de fecha. Con este formato, recuerde que el siglo por defecto es el actual. Como la fecha también contiene información de hora, la hora por defecto es la medianoche (00:00:00).

Si se debe introducir una fecha en un formato distinto del formato por defecto, por ejemplo, con otro siglo, o una hora específica, debe utilizar la función TO_DATE.

En el ejemplo se registra información para el empleado Raphealy en la tabla EMPLOYEES y se define la columna HIRE_DATE en 3 de febrero de 1999. Si utiliza la siguiente sentencia en lugar de la mostrada en la transparencia, el año de HIRE_DATE se interpreta como 2099.

```
I NSERT I NTO emp loyees
VALUES          (114 , 'Den ' ,  'Raphea l y ' ,
    'DRAPHEAL',   ' 515 .126 .4561 ' ,
    '03-FEB-99' ,'AC_ACCOUNT' ,
      11000 , NULL , 100 , 30 ) ;
```

7.2.2. Creación de un Archivo de Comandos

Puede guardar comandos con variables de sustitución en un archivo y ejecutar los comandos del archivo. En el ejemplo anterior se registra información para un departamento en la tabla DEPARTMENTS.

Ejecute el archivo de comandos y se le pedirá que introduzca las variables de sustitución &. Los valores que introduzca se sustituirán más tarde en la sentencia. Esto le permite ejecutar el mismo archivo de comandos una y otra vez, suministrando un juego de valores distinto cada vez que lo ejecute.

```
INSERT INTO depa r tmen t s ( depa r t men t _ i d ,
                    depa r tmen t _name , l oca t ion_ i d )
VALUES ( &depa r tmen t_ i d , &depa r t men t _name ,
            & l oca t ion_ id )
```

7.2.3. Copia de Filas de Otra Tabla

Puede utilizar la sentencia INSERT para agregar filas a una tabla en la que los valores se derivan desde tablas existentes. En lugar de la cláusula VALÚES, utilice una subconsulta.

Sintaxis

INSERT INTO table [column (, column)] subquery;

En la sintaxis:

table es el nombre de la tabla.

column es el nombre de la columna de la tabla que se ha de rellenar.

subquery es la subconsulta que devuelve filas a la tabla.

El número de columnas y sus tipos de dato en la lista de columnas de la cláusula INSERT debe coincidir con el número de valores y sus tipos de dato en la subconsulta. Para crear una copia de las filas de una tabla, utilice SELECT * en la subconsulta.

```
I NSERT I NTO copy_emp
SELECT * FROM    emp l oyees ;
```

7.2.4. Uso de una Subconsulta en una Sentencia INSERT

Puede utilizar una Subconsulta en lugar del nombre de tabla en la cláusula INTO de la sentencia INSERT.

La lista de selección de esta subconsulta debe tener el mismo número de columnas que la lista de columnas de la cláusula VALÚES. Se deben seguir todas las reglas de las columnas de la tabla base para que la sentencia INSERT funcione correctamente. Por ejemplo, no puede poner un identificador de empleado duplicado, ni dejar fuera un valor para una columna obligatoria no nula.

```
I NSERT I NTO ( SELECT emp loyee_ id , las t_name ,
                                email , hire_date, job_ i d, salary,
                                depar tment id FROM    emp loyees
            WHERE department_id =50)
VALES (99999, 'Tay lor', 'DTAYLOR',
TO_DATE ( ' 06 -JUN -99 ' , 'DD - MON -RR ' ) , •ST_CLERK ' , 5000 , 50 ) ;
```

7.2.5. Uso de las Palabras Clave WITH CHECK OPTION en

Sentencias DML

Especifique WITH CHECK OPTION para indicar que, si se utiliza la subconsulta en lugar de una tabla en una sentencia INSERT, UPDATE o ÓBLETE, en la tabla no se permitirán cambios que produzcan filas que no estén incluidas en la subconsulta.

En el ejemplo mostrado, se utiliza la palabra clave WITH CHECK OPTION. La subconsulta identifica las filas que están en el departamento 50, pero el identificador de

departamento no se encuentra en la lista SELECT y no se le proporciona un valor en la lista VALÚES. La inserción de esta fila daría como resultado un identificador de departamento de valor nulo, que no se encuentra en la subconsulta.

```
INSERT INTO    (SELECT employee_id, last_name, email,
   hire_date, job_id, salary
   FROH    employees
   WHERE   department_id = 50 WITH CHECK OPTION)
 VALUES (99998, 'Smith', 'JSMITH',TO_DATE('06-JUN-99', 'DD-
MON-RR'), 'ST_CLERK', 5000);
ERROR at line 1 :
ORA -01402 : view WITH CHECK OPTION where -clause violation
```

7.2.6. Sintaxis de la Sentencia UPDATE

Puede modificar las filas existentes utilizando la sentencia UPDATE. En la sintaxis:

table es el nombre de la tabla.

column es el nombre de la columna de la tabla que se ha de rellenar.

value es la subconsulta o el valor correspondiente para la columna.

condition identifica las filas que se han de actualizar y está compuesta de expresiones de nombres de columna, constantes, subconsultas y operadores de comparación.

Confirme la operación de actualización mediante la consulta a la tabla para mostrar las filas actualizadas.

Nota: En general, utilice la clave primaria para identificar una sola fila. Si usa otras columnas se pueden actualizar varias filas de forma inesperada. Por ejemplo, identificar una sola fila de la tabla EMPLOYEES por el nombre es peligroso, ya que es posible que varios empleados tengan el mismo nombre.

```
UPDATE      tabla
SET column = value [ , column = value , . . .]
WHERE       condition];
```

7.2.7. Actualización de las Filas de una Tabla

La sentencia UPDATE modifica filas específicas si se incluye la cláusula WHERE. En el ejemplo se transfiere al empleado 113 (Popp) al departamento 60.

Nota: Si omite la cláusula WHERE, se modifican todas las filas de la tabla.

```
UPDATE emp l oyees SET depa r tmen t_ id = 60
WHERE emp loyee_ id=113
```

7.2.8. Actualización de Dos Columnas con una Subconsulta

Puede actualizar varias columnas en la cláusula SET de una sentencia UPDATE escribiendo varias subconsultas.

Sintaxis

```
UPDATE tab le  SET co l umn  = (SELECT co l umn
FROM tab l e
WHERE cond i t i on )
  co l umn= (SELECT  co l umn  FROM  tab le  WHERE  cond i t ion)
        [WHERE    condi t i on  ]          ;
```

7.2.9. Actualización de Filas Basándose en Otra Tabla

Puede utilizar subconsultas en sentencias UPDATE para actualizar las filas de una tabla. En el ejemplo se actualiza la tabla COPY_EMP basándose en los valores de la tabla EMPLOYEES, se cambia el número de departamento de todos los empleados con el identificador de cargo del empleado 200 por el número de departamento actual del empleado 100.

```
UPDATE copy_emp  SET  department_id=(SELECT  depar tment_id
      FROM emp loyees
      WHERE emp loyee_ i d=200 )
    WHERE job_ id = (SELECT job_ i d
       FROM emp loyees_ i d
```

```
WHERE emp loyee_ id=200 )
```

7.2.10. Sentencia DELETE

Puede eliminar las filas existentes utilizando la sentencia DELETE.

En la sintaxis:

table es el nombre de la tabla

condition identifica las filas que se van a suprimir y está compuesta de nombres de columna, expresiones, constantes, subconsultas y operadores decomparación.

```
DELETE [ FROM]    tab le
[WHERE                cond i t i on ] ;
```

Nota: Si no se suprime ninguna fila, se devuelve el mensaje "O rows deleted".

7.2.11. Supresión de Filas de una Tabla

Puede suprimir filas específicas incluyendo la cláusula WHERE en la sentencia DELETE. En el ejemplo se suprime el departamento Finance de la tabla DEPARTMENTS.

Si omite la cláusula WHERE, todas las filas de la tabla se suprimen.

```
DELETE FROM emp l oyees
WHERE depa r t men t _name= 'F I NANCE '
```

Ejemplo

Elimine las filas identificadas en la cláusula WHERE.

```
DELETE FROM emp loyees
WHERE          emp l oyee_ i d = 114 ;
```

7.2.12. Supresión de Filas Basándose en Otra Tabla

Puede utilizar subconsultas para suprimir filas de una tabla basándose en valores de otra tabla. En el ejemplo se suprimen todos los empleados que están en un departamento cuyo nombre contiene la cadena "Public".

La subconsulta busca en la tabla DEPARTMENTS el número de departamento basándose en el nombre de departamento que contiene la cadena "Public". A continuación, proporciona el número de departamento a la consulta principal, la cual suprime filas de datos de la tabla EMPLOYEES basándose en este número de departamento.

```
DELETE FROM emp loyees
WHERE depa r tmen t_ id = (SELECT depa r tmen t_ i d
                    FROM    depa r tmen ts
                    WHERE   depa r tmen t_name  L IKE '%Pub l ic% '
```

7.2.13. Sentencia TRUNCATE

Efectos de la sentencia TRUNCATE:

Se suprimen todas las filas de la tabla

No se generan segmentos de UNDO y el comando se valida automáticamente porque es un comando DDL (Data Definición Language).

También se truncan los indices correspondientes

No se puede truncar una tabla que hace referencia a una FOREING KEY

Los TRIGGERS de borrado no arrancan cuando se utiliza este comando

A diferencia del comando DELETE que si es DML libera espacio al borrar la tabla

La sintaxis es:

```
TRUNCATE TABLE employees
```

7.2.14. Visión General de la Función Valor por Defecto Explícito

Con la función valor por defecto explícito, puede utilizar la palabra clave DEFAULT como valor de columna donde desee el valor por defecto de columna.

Esta función se agrega para asegurar el cumplimiento con el estándar SQL: 1999.

Esto permite al usuario controlar dónde y cuándo se debe aplicar el valor por defecto a los datos.

Se pueden utilizar valores por defecto explícitos en sentencias INSERT y UPDATE.

Especifique DEFAULT para definir la columna en el valor especificado previamente como valor por defecto para la columna. Si no se ha especificado ningún valor por defecto para la columna correspondiente, Oracle define la columna en un valor nulo.

En el primer ejemplo mostrado, la sentencia INSERT utiliza un valor por defecto para la columna MANAGER_ID. Si no hay ningún valor por defecto definido para la columna, se inserta en su lugar un valor nulo.

En el segundo ejemplo se utiliza la sentencia UPDATE para definir la columna MANAGER_ID en un valor por defecto para el departamento 10. Si no hay ningún valor por defecto definido para la columna, cambia el valor a nulo.

Nota: Al crear una tabla, puede especificar un valor por defecto para una columna

```
DEFAULT CON I NSERT :
 INSERT INTO departments (department_id, depar tment_name,
manager_ i d)
VALUES (300 , ' Eng inee r ing ' , DEFAULT ) ;
```

DEFAULT con UPDATE:

```
    UPDATE emp loyees SET manage r_ id=DEFAULT
WHERE depa r tmen t_ i d=10
```

7.2.15. Sentencias MERGE

SQL se ha extendido para incluir la sentencia MERGE. Con esta sentencia, puede actualizar o insertar una fila condicionalmente en una tabla, evitando así tener múltiples sentencias UPDATE. La decisión sobre si actualizar o insertar en la tabla de destino se basa en una condición en la cláusula ON.

Como el comando MERGE combina los comandos INSERT y UPDATE, necesita los dos privilegios INSERT y UPDATE en la tabla de destino y el privilegio SELECT en la tabla de origen.

La sentencia MERGE es determinista. No puede actualizar la misma fila de la tabla de destino varias veces en la misma sentencia MERGE.

Un enfoque alternativo es utilizar bucles PL/SQL y varias sentencias DML. Sin embargo, la sentencia MERGE es fácil de utilizar y se expresa de forma más simple como una sola sentencia SQL.

La sentencia MERGE es útil en diversas aplicaciones de almacenes de datos. Por ejemplo, en una aplicación de almacenes de datos, tal vez sea necesario trabajar con datos procedentes de varios orígenes, algunos de los cuales pueden estar duplicados. Con la sentencia MERGE, puede agregar o modificar filas condicionalmente.

7.2.16. Sintaxis de la Sentencia MERGE

MERGE INTO table_name table_aliaa USING (table/viewjsub_query) alias ON (join condítionl WHEN MATCHED THEN

```
ÜPDATE SET
co ll = co l_va l l ,
co l  2  =  co l 2_va l WHEN NOT MATCHED THEN
I NSERT ( co l umn_ l is t)
VALÚES       ( co lumn va l úes) ;
```

Puede actualizar filas existentes e insertar filas nuevas condicionalmente utilizando la sentencia MERGE.

En la sintaxis:

Cláusula INTO especifica la tabla de destino que está actualizando o en laque está insertando

Cláusula USING identifica el origen de los datos que se van a actualizar o insertar, puede ser una tabla, una vista o una subconsulta

Cláusula ON la condición sobre la cual la operación MERGE actualiza o inserta WHEN MATCHED indica al servidor cómo responder a los resultados de la condición WHEN NOT de unión.MATCHED

7.2.17. Transacciones de Base de Datos

Oracle Server asegura la consistencia de datos basándose en transacciones.

Las transacciones le ofrecen más flexibilidad y control al cambiar datos y aseguran una consistencia de datos en caso de fallo del proceso de usuario o fallo del sistema.

Las transacciones constan de sentencias DML que constituyen un cambio consistente en los datos.

Por ejemplo, una transferencia de fondos entre dos cuentas debe incluir el débito a una cuenta y el cargo a otra por el mismo importe. Las dos acciones se deben realizar correcta o incorrectamente al mismo tiempo, de forma que el crédito no se debe validar sin el débito.

Una transacción comienza cuando se encuentra la primera sentencia DML y termina cuando se produce una de las siguientes opciones:

- Se emite una sentencia COMMIT o ROLLBACK.
- Se emite una sentencia DDL, como por ejemplo, CRÉATE.
- Se emite una sentencia DCL.
- El usuario sale de la sesión.
- Se produce un fallo de una máquina o en el sistema.

Cuando termina un transacción, la siguiente sentencia SQL ejecutable inicia automáticamente la siguiente transacción.

Se valida automáticamente una sentencia DDL o una sentencia DCL y, por lo tanto, finaliza implícitamente una transacción.

7.2.18. Ventajas de las Sentencias COMMIT y ROLLBACK

Con sentencias commit y rollback, puede:

- Asegurar la consistencia de datos

- Realizar una presentación preliminar de los cambios de datos antes de hacer que estos sean permanentes

- Agrupar las operaciones relacionadas lógicamente

7.2.19. Sentencias Explícitas de Control de Transacciones

Puede controlar la lógica de las transacciones utilizando las sentencias COMMIT, SAVEPOINT y ROLLBACK

Sentencia	Descripción
COMMIT	Termina la transacción actual haciendo permanentes todos los cambios de datos pendientes.
SAVEPOINT name	Marca un punto de grabación dentro de la transacción actual.
ROLLBACK	ROLLBACK termina la transacción actual desechando todos los cambios de datos pendientes.
ROLLBACK TO SAVEPOINT name	ROLLBACK TO SAVEPOINT realiza rollback de la transacción actual al punto de grabación especificado, desechando así los cambios o los puntos de grabación creados después del punto de grabación al que está realizando rollback. Si omite la cláusula TO SAVEPOINT, la sentencia ROLLBACK realiza rollback de toda la transacción. Como los puntos de grabación son lógicos, no hay ninguna forma de enumerar los puntos de grabación que ha creado.

Nota: SAVEPOINT no es SQL del estándar ANSÍ.

7.2.20. Rollback a un Marcador

Puede crear un marcador en la transacción actual utilizando la sentencia SAVEPOINT, que divide la transacción en secciones más pequeñas. A continuación, puede desechar los cambios pendientes hasta dicho marcador utilizando la sentencia ROLLBACK TO SAVEPOINT.

Si crea un segundo punto de grabación con el mismo nombre que uno anterior, el

primero se suprime.

```
UPDATE. . .
AVEPO INT upda t e_done
Savepo in t c rea t ed .
I NSERT . . .
ROLLBACK TO upda t e_done ;
Ro l lback comp l e t e .
```

7.2.21. Procesamiento Implícito de Transacciones

Nota: En 'SQL*Plus está disponible un tercer comando. El comando AUTOCOMMIT se puede activar o desactivar. Si está activado, cada sentencia DML individual se valida en cuanto se ejecuta.

No puede realizar rollback de los cambios. Si está desactivado, la sentencia COMMIT todavía se puede emitir explícitamente.

Además, la sentencia COMMIT se emite cuando se emite una sentencia DDL o al salir de la sesión .

Fallos del Sistema

Cuando se interrumpe una transacción por un fallo del sistema, se realiza rollback automáticamente a toda la transacción.

Esto evita que el error produzca cambios no deseados en los datos y devuelve las tablas al estado que tenían en el momento de la última validación. De esta forma, Oracle Server protege la integridad de las tablas.

Con SQL*Plus, se consigue salir normalmente escribiendo el comando EXIT en el prompt. Cerrar la ventana se interpreta como una salida anormal.

7.2.22. Validación de Cambios

Todo cambio de datos realizado durante la transacción es temporal hasta que se valida la transacción.

Estado de los datos antes de que se emitan sentencias COMMIT o ROLLBACK:

Las operaciones de manipulación de datos afectan principalmente al buffer de la base de datos; por lo tanto, el estado anterior de los datos se puede recuperar.

El usuario actual puede revisar los resultados de las operaciones de manipulación de datos consultando las tablas.

Otros usuarios no pueden ver los resultados de las operaciones de manipulación de datos realizados por el usuario actual. Oracle Server establece la consistencia de lectura para asegurar que cada usuario ve los datos como eran en la última validación.

Las filas afectadas se bloquean; otros usuarios no pueden cambiar los datos de las filas afectadas

Haga que todos los cambios pendientes sean permanentes utilizando la sentencia COMMIT. Tras una sentencia COMMIT:

los cambios de datos se escriben en la base de datos.

El estado anterior de los datos se pierde permanentemente.

Todos los usuarios pueden ver los resultados de la transacción.

Los bloqueos de las filas afectadas se liberan; las filas están ahora disponibles para que otros usuarios realicen nuevos cambios de datos.

Todos los puntos de grabación se borran.

En el ejemplo se suprime una fila de la tabla EMPLOYEES y se inserta una fila nueva en la tabla DEPARTMENTS.

A continuación, se hace que el cambio sea permanente emitiendo la sentencia COMMIT

Ejemplo

Elimine los departamentos 290 y 300 de la tabla DEPARTMENTS y actualice una fila en la tabla COPY_EMP. Haga que el cambio de datos sea permanente.

```
DELETE FROM depa r tmen ts
WHERE depa r tmen t_ id I N (290 , 300 ) ;
2 r ows de le ted .
UPDATE copy_emp
SET     depa r tmen t_ i d =80 WHERE emp loyee_ i d = 206;
1 r ow upda t ed .
COMM I T ;
```

```
Comm i t Comp1e te .
```

7.2.23. Rollback de los cambios

Deseche todos los cambios pendientes utilizando la sentencia ROLLBACK. Tras una sentencia ROLLBACK:

Los cambios de datos se deshacen. ,

El estado anterior de los datos se restaura.

Los bloqueos de las filas afectadas se liberan. Ejemplo

Al intentar eliminar un registro de la tabla TEST, puede vaciar accidentalmente la tabla. Puede corregir el error, volver a emitir la sentencia correcta y hacer que el cambio de datos sea permanente.

```
DELETE FROM t es t ;
25 , 000 r ows de l e ted .
ROLLBACK ;
Ro l lback comp l e t e .
DELETE FROM t es t WHERE     i d =  100 ;
1 r ow de l e t ed .
SELECT  * FROM       tes t  WHERE     i d =  100 ;
No rows se l ec ted .
COMM I T ;
Comm i t comp le te .
```

7.2.24. Rollbacks a Nivel de Sentencia

Un rollback implícito puede desechar parte de una transacción si se detecta un error de ejecución de sentencia. Si falla una sola sentencia DML durante la ejecución de una transacción, su efecto lo deshace un rollback a nivel de sentencia, pero los cambios realizados por las sentencias DML anteriores en la transacción no se desechan. El usuario puede validarlas o realizar rollback de ellas explícitamente.

Oracle emite una validación implícita antes y después de cualquier sentencia del lenguaje de definición de datos (DDL). Por ello, aunque la sentencia DDL no se ejecute

correctamente, no puede realizar rollback de la sentencia anterior porque el servidor emitió una validación.

Termine las transacciones explícitamente ejecutando una sentencia COMMIT o ROLLBACK.

7.2.25. Consistencia de Lectura

Los usuarios de la base de datos acceden a ésta de dos formas:

Operaciones de lectura (sentencia SELECT)

Operaciones de escritura (sentencias INSERÍ, UPDATE, DELETE) Necesita consistencia de lectura para que:

El escritor y el lector de la base de datos se aseguren una visualización consistente de los datos.

Los lectores no visualicen los datos que están en proceso de cambio.

Los escritores se aseguren de que los cambios en la base de datos se realizan de forma consistente.

Los cambios realizados por un escritor no interrumpan a los que está haciendo otro escritor ni entren en conflicto con ellos.

El objetivo de la consistencia de lectura es asegurar que cada usuario ve los datos tal como eran en la última validación, antes de que comenzara una operación DML.

7.2.26. Implementación de Consistencia de Lectura

La consistencia de lectura es una implementación automática. Mantiene una copia parcial de la base de datos en segmentos deshacer.

Cuando se realiza una operación de inserción, actualización o supresión en la base de datos, Oracle Server realiza una copia de los datos antes de que se cambien y los escribe en un segmento deshacer.

Todos los lectores, excepto el que emitió el cambio, seguirán viendo la base de datos como existía antes de que comenzaran los cambios; ven la "instantánea" del segmento de rollback de los datos.

Antes de que se validen los cambios en la base de datos, sólo el usuario que está modificando los datos ve la base de datos con las modificaciones; los demás ven la instantánea en el segmento deshacer. Esto garantiza que los lectores de los datos leen datos consistentes que no están sufriendo ningún cambio actualmente.

Cuando se valida una sentencia DML, el cambio realizado en la base de datos se hace visible para cualquier persona que ejecute una sentencia SELECT. El espacio ocupado por los datos antiguos en el archivo del segmento deshacer se libera para su nuevo uso. Si se realiza rollback de la transacción, los cambios se deshacen:

La versión original (más antigua) de los datos del segmento deshacer se vuelve a escribir en la tabla.

Todos los usuarios ven la base de datos como existía antes de que comenzara la transacción.

7.2.27. Bloqueos

Los bloqueos son mecanismos que evitan una interacción destructiva entre las transacciones que acceden al mismo recurso, ya sea un objeto de usuario (como tablas o filas) o un objeto de sistema no visible a los usuarios (como estructuras de datos compartidos y filas del diccionario de datos).

Cómo Bloquea Datos la Base de Datos Oracle

El bloqueo de Oracle se realiza automáticamente y no requiere acción del usuario. El bloqueo implícito se produce para las sentencias SQL según sea necesario, dependiendo de la acción solicitada. Este bloqueo se produce para todas las sentencias SQL excepto SELECT.

Los usuarios también pueden bloquear los datos manualmente, lo que se denomina bloqueo explícito.

7.2.28. Bloqueo Implícito

Al realizar operaciones del lenguaje de manipulación de datos (DML), Oracle Server proporciona simultaneidad de datos a través del bloqueo DML, que se produce a dos niveles:

Un bloqueo compartido se obtiene automáticamente a nivel de tabla durante operaciones DML: Con el modo de bloqueo compartido, varias transacciones pueden adquirir bloqueos compartidos sobre el mismo recurso.

Un bloqueo exclusivo se adquiere automáticamente para cada fila modificada por una sentencia DML. Los bloqueos exclusivos evitan que otras transacciones cambien la fila hasta que la transacción se valida o se realiza rollback. Este bloqueo asegura que ningún otro usuario puede modificar la misma fila al mismo tiempo ni sobrescribir los cambios que aún no han sido validados por otro usuario.

Los bloqueos DDL se producen al modificar un objeto de base de datos como una tabla.

7.2.29. Sentencias INSERT de Varias Tablas

En una sentencia INSERT de varias tablas, se insertan filas calculadas derivadas de las filas devueltas de la evaluación de una subconsulta en una o más tablas.

Las sentencias INSERT de varias tablas pueden desempeñar un papel muy útil en el supuesto de un almacén de datos. Debe cargar el almacén de datos con regularidad para que pueda cumplir su propósito de facilitar el análisis de negocio. Para ello, se deben extraer y copiar datos de uno o más sistemas operativos al almacén de datos. El proceso de extracción de datos del sistema de origen y su transferencia al almacén de datos se suele denominar ETL (siglas de extraction, transformation, and loading, o extracción, transformación y carga).

Durante la extracción, se deben identificar y extraer los datos deseados de diferentes orígenes como, por ejemplo, aplicaciones y sistemas de bases de datos. Después de la extracción, los datos se deben transportar físicamente al sistema de destino o a un sistema intermedio para continuar su procesamiento. Dependiendo del medio de transporte seleccionado, algunas transformaciones se pueden realizar durante este proceso. Por ejemplo, una sentencia SQL que acceda directamente a un destino remoto

a través de un gateway puede concatenar dos columnas como parte de la sentencia SELECT.

Una vez cargados los datos en la base de datos Oracle, las transformaciones de datos se pueden ejecutar mediante operaciones SQL. Una sentencia INSERT de varias tablas es una de las técnicas para implementar transformaciones de datos SQL.

Sintaxis

```
I NSERT ALL
INTO table_a VALUES(...,...,...)
INTO table_b VALUES(...,...,...)
INTO table_c VALUES(...,...,...)
SELECT ...
FROM sou r ce tab
WHERE ... ;
```

Las sentencias INSERT de varias tablas ofrecen las ventajas de la sentencia INSERT ... SELECT cuando hay varias tablas implicadas como destinos. Con la funcionalidad anterior a la base de datos Oracle9i, era necesario tratar con n sentencias INSERT ... SELECT independientes, procesando los mismos datos de origen n veces y aumentando la carga de trabajo de transformación n veces.

Como sucede con la sentencia INSERT ... SELECT existente, la nueva sentencia se puede paralelizar y utilizar con el mecanismo de carga directa para obtener un rendimiento más rápido.

Cada registro de cualquier flujo de entrada como, por ejemplo, una tabla de base de datos no relacional, se puede convertir ahora en varios registros para un entorno de tabla de base de datos más relacional. Para implementar esta funcionalidad de forma alternativa, había que escribir varias sentencias INSERT.

7.2.30. Tipos de Sentencias INSERT de Varias Tablas

Los tipos de sentencias INSERT de varias tablas son:

- I NSERT incond i c i ona l
- ALL INSERT cond i c iona l
- F IRST I NSERT cond ic ional
- I NSERT de p ivo t i ng

Se utilizan diferentes cláusulas para indicar el tipo de inserción (INSERT) que se ejecutará.

7.2.31. INSERT ALL Incondicional

El ejemplo inserta filas en las tablas SAL_HISTORY y MGR_HISTORY.

La sentencia SELECT recupera los detalles de identificador de empleado, fecha de contratación, salario e identificador de supervisor de los empleados cuyo identificador de empleado es mayor que 200 en la tabla EMPLOYEES. Los detalles de identificador de empleado, fecha de contratación y salario se insertan en la tabla SAL_HISTORY. Los detalles de identificador de empleado, identificador de supervisor y salario se insertan en la tabla MGR_HISTORY.

La sentencia INSERT se conoce como INSERT incondicional, ya que no se aplican más restricciones a las filas que se recuperan mediante la sentencia SELECT. Todas las filas recuperadas mediante la sentencia SELECT se insertan en las dos tablas, SAL_HISTORY y MGR_HISTORY. La cláusula VALUES de las sentencias INSERT especifica las columnas de la sentencia SELECT que se deben insertar en cada una de las tablas. Cada fila devuelta mediante la sentencia SELECT da como resultado dos inserciones, una para la tabla

SAL_HISTORY y una para la tabla MGR_HISTORY.

Se puede interpretar que el feedback 8 rows created significa que se realizó un total de ocho inserciones en las tablas base, SAL_HISTORY y MGR_HISTORY.

```
INSERT      ALL
INTO sal_history VALUES(EMPID,HIREDATE,SAL) INTO mgr_history
VALUES(EMPID,MGR,SAL)
SELECT emp l oyee_ id EMP ID , h i re_da te H IREDATE , sa la ry SAL ,
manage r_ i d MGR
FROM      emp loyees
WHERE emp loyee_ id > 200 ;
8 r ows c r ea ted .
```

7.2.32. INSERT ALL Condicional

El ejemplo es parecido al anterior, ya que inserta filas en las tablas SAL_HISTORY y MGR_HISTORY. La sentencia SELECT recupera los detalles de identificador de empleado, fecha de contratación, salario e identificador de supervisor de los empleados cuyo identificador de empleado es mayor que 200 en la tabla EMPLOYEES. Los detalles de identificador de empleado, fecha de contratación y salario se insertan en la tabla SAL_HISTORY. Los detalles de identificador de empleado, identificador de supervisor y salario se insertan en la tabla MGR_HISTORY.

La sentencia INSERT se conoce como ALL INSERT condicional, ya que se aplica una restricción más a las filas que se recuperan mediante la sentencia SELECT. De las filas recuperadas mediante la sentencia SELECT, sólo aquéllas en las que el valor de la columna

SAL sea mayor que 10000 se insertarán en la tabla SAL_HISTORY y, de forma parecida, sólo las filas en las que el valor de la columna MGR sean mayor que 200 se insertarán en la tabla MGR_HISTORY.

Observe que, a diferencia del ejemplo anterior, en el que se insertaron ocho filas en las tablas, en este ejemplo sólo se insertan cuatro filas.

Se puede interpretar que el feedback 4 rows created significa que se realizó un total de cuatro inserciones en las tablas base, SAL_HISTORY y MGR_HISTORY.

```
INSERT ALL
WHEN        SAL > 10000 THEN
INTO sal_history VALUES(EMPID,HIREDATE,SAL)
WHEN        MGR  >  200          THEN
INTO mgr_history VALUES (EMPID,MGR,SAL)
SELECT employee_id  EMPID, hire_date  HIREDATE, salary  SAL,
manager_id MGR
FROM         employees
WHERE      employee_ id > 200;
4 rows created
```

7.2.33. INSERT FIRST Condicional

El ejemplo inserta filas en más de una tabla mediante una única sentencia INSERT.

La sentencia SELECT recupera los detalles de identificador de departamento, salario total y fecha de contratación máxima de cada departamento de la tabla EMPLOYEES.

Esta sentencia INSERT se conoce como FIRST INSERT condicional, ya que se realiza una excepción para los departamentos cuyo salario total sea mayor que 25.000 pesos. La condición WHEN ALL > 25000 se evalúa en primer lugar.

Si el salario total de un departamento es mayor que 25.000 dólares, el registro se inserta en la tabla SPECIAL_SAL, independientemente de la fecha de contratación.

Si esta primera cláusula WHEN se evalúa como verdadera, Oracle Server ejecuta la cláusula INTO correspondiente y salta las cláusulas WHEN siguientes para esta fila.

Para las filas que no satisfacen la primera condición WHEN (WHEN SAL > 25000), el resto de las condiciones se evalúa igual que una sentencia INSERT condicional y los registros recuperados mediante la sentencia SELECT se insertan en las tablas HIREDATE_HISTORY_ 00 o HIREDATE_HISTORY_99 o HIREDATE_HISTORY, basándose en el valor de la columna HIREDATE.

Se puede interpretar que el feedback 8 rows created significa que se realizó un total de ocho sentencias INSERT en las tablas base, SPECIAL_SAL, HIREDATE_HISTORY_00, HIREDATE_HISTORY_99 y HIREDATE_HISTORY.

```
INSERT      FIRST
WHEN SAL  > 25000THEN
      INTO special_sal VALUES(DEPTID, SAL)
  WHEN HIREDATE like ('%00%') THEN
INTO hiredate_history_00 VALUES(DEPTID,HIREDATE)
            WHEN HIREDATE like ('%99%') THEN
INTO hiredate_history_99 VALUES(DEPTID, HIREDATE)
ELSE
              INTO hiredate_history VALUES(DEPTID, HIREDATE)
SELECT department_id DEPTID, SUM(salary) SAL,
              MAX(hire_date) HIREDATE
FROM    employees
GROUP BY depar tmen t _ id ;
7      rows c r ea t ed .
```

7.2.34. INSERT de Pivoting

El pivoting es una operación en la que se debe crear una transformación tal que cada registro de cualquier flujo de entrada como, por ejemplo, una tabla de base de datos no relacional, se debe convertir en varios registros para un entorno de tablas de base de datos más relacional.

Para solucionar el problema que se menciona en la diapositiva, debe crear una transformación tal que cada registro de la tabla de base de datos no relacional original, SALES_SOURCE_ DATA, se convierta en cinco registros para la tabla SALES_INFO del almacén de datos. Esta operación se suele conocer como pivoting.

En la diapositiva se especifica la sentencia con un problema para una sentencia INSERT de pivoting. La solución a este problema se muestra en la página siguiente.

```
INSERT ALL
INTO sales_info VALUES (employee_id,week_id,sales_MON)
INTO sales_info VALUES (employee_id,week_id,sales_TUE)
INTO sales_info VALUES (employee_id,week_id,sales_WED)
INTO sales_info VALUES (employee_id,week_id,sales_THUR)
INTO sales_info VALUES (employee_id,week_id, sales_FRI)
SELECT employee_id, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR,sales_FRI
FROM sales_source_data;
5 rows created.
```

7.2.35. Seguimiento de Cambios en los Datos

Puede suceder que, de algún modo, haya datos en la tabla que se hayan cambiado incorrectamente.

Para investigar esto, puede utilizar varias consultas de flashback para ver los datos de filas en puntos concretos en el tiempo. Lo que es más eficaz, puede utilizar la función Consulta de Versiones de Flashback para ver todos los cambios efectuados en una fila durante un período de tiempo. Esta función le permite agregar una cláusula VERSIONS a

una sentencia SELECT que especifique un SCN o rango de registro de hora en el que desee ver cambios en los valores de fila.

La consulta también puede devolver metadatos asociados como, por ejemplo, la transacción responsable del cambio.

Es más, después de identificar una transacción errónea, puede utilizar la función Consulta de Versiones de Flashback para identificar otros cambios que haya realizado la transacción. Puede utilizar entonces la función Consulta de Versiones de Flashback para restaurar la tabla a un estado anterior a la realización de los cambios.

Puede utilizar una consulta en una tabla con una cláusula VERSIONS para producir todas las versiones de todas las filas que existen o que hayan existido entre el momento en que se emitió la consulta y los segundos undo_retention anteriores al momento actual. undo_retention es un parámetro de inicialización de ajuste automático.

Una consulta que incluye una cláusula VERSIONS se conoce como consulta de versiones.

Los resultados de una consulta de versiones se comportan como si se hubiera aplicado la cláusula WHERE a las versiones de las filas. La consulta de versiones devuelve versiones de las filas sólo en transacciones.

SCN (número de cambio del sistema): Oracle Server asigna un SCN (número de cambio del sistema) para identificar los registros de rehacer para cada transacción validada.

7.2.36. Ejemplo de Consulta de Versiones de Flashback

Ejemplo , recuperar el salario del empleado 106.

```
SELECT sa la ry FROM emp loyees3
WHERE      emp loyee_ id = 106 ;
E l sa l a r io  de l  emp l eado  106  se  aumen t a  en  un  30  po r  c ien t o  y
se va li da e l camb io .
                                    UPDATE emp loyees3 SET sa la r y = sa l a r y *
1 .30
                WHERE      emp loyee_ i d = 106;
               COMM I T;
```

Se muestran las diferentes versiones del salario.

```
SELECT sa la ry FROM emp loyees3
```

VERS IONS BETWEEN SCN M I NVALUE AND MAXVALUE

```
WHERE      emp loyee_ id = 106;
```

La cláusula VERSIONS no cambia el plan de la consulta.

Por ejemplo, si ejecuta una consulta en una tabla que utilice el método de acceso a índices, la misma consulta de la misma tabla con una cláusula VERSIONS continúa utilizando el método de acceso a índices.

Las versiones de las filas devueltas por la consulta de versiones son versiones de las filas en las transacciones.

La cláusula VERSIONS no afecta al comportamiento transaccional de una consulta. Esto significa que una consulta en una tabla con una cláusula VERSIONS sigue heredando el entorno de la consulta de la transacción en curso.

La cláusula VERSIONS por defecto se puede especificar como VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE.

La cláusula VERSIONS es una extensión SQL únicamente para consultas.

Puede tener operaciones DML y DDL que utilicen una cláusula VERSIONS dentro de las subconsultas.

La consulta de versiones recupera todas las versiones validadas de las filas seleccionadas.

Los cambios realizados por la transacción activa actual no se devuelven.

La consulta de versiones recupera todas las encarnaciones de las filas.

Esto significa en esencia que las versiones devueltas incluyen las versiones suprimidas y subsiguientes reinsertadas de las filas.

El acceso a filas para una consulta de versiones se puede definir en una de las siguientes dos categorías:

Acceso a filas basado en ROWID: En el caso del acceso basado en ROWID, se devuelven todas las versiones de los ROWID independientemente del contenido de las filas.

Básicamente, esto significa que se devuelven todas las versiones de la ranura del bloque indicado por el ROWID.

Acceso a todas las demás filas: Para el acceso a todas las demás filas, se devuelven todas las versiones de las filas

7.2.37. Cláusula VERSIONS BETWEEN

Puede utilizar la cláusula VERSIONS BETWEEN para recuperar todas las versiones de las filas que existen o que han existido entre el momento en que se emitió la consulta y un punto pasado en el tiempo.

Si el momento de retención de deshacer es mejor que el momento de límite inferior/SCN de la cláusula BETWEEN, la consulta recupera únicamente las versiones hasta el momento de retención de deshacer. El intervalo de tiempo de la cláusula BETWEEN se puede especificar como intervalo SCN o como intervalo de reloj. Este intervalo de tiempo se cierra en los límites inferior y superior.

En el ejemplo, se recuperan los cambios de salario de Lorentz. El valor NULL para END_DATE para la primera versión indica que se trataba de la versión existente en el momento de la consulta. El valor NULL para START_DATE para la última versión indica que esta versión se creó en un momento anterior a la retención de deshacer.

```
SELECT versions_starttime "START_DATE",

      versions_endtime     "END_DATE",

                           sa l a ry
         FROM              emp loyees
         VERS IONS BETWEEN SCN M INVALUE AND MAXVALUE
         WHERE       l as t_name =  ' Lo ren tz ' ;
```

Ejercicios

Para realizar estos ejercicios corra el script mod_vii

Salvo indicación contraria en todos los casos confirmar la transacción y visualizar los cambios

1. Cargar la tabla EMP_TOT con los empleados de la empresa sin incluir los del departamento 80.

2. Actualizar el sueldo de los empleados del departamento 30 en un 35%

3. Eliminar los empleados del departamento 10

4. Cargue los empleados de la empresa cuyo sueldo sea superior a $10.000 en la tabla EMP_SSUP. Los que tengan un sueldo inferior cargarlos en la tabla EMP_SINF

Utilizar Insert condicional.

5. Actualice el salario del empleado cuyo legajo es el número 200 con un 40% adicional.

Confirmar la operación. Deshacer el cambio usando instrucciones flash back.

8. Gestión de tablas

8.1. Objetivos

Al finalizar la lección el alumno estará en condiciones de crear tablas de distintos tipos soportados por ORACLE, modificarlas y borrarlas.

8.1.1. Objetos de Base de Datos

Objeto	Descripción
Tabla	Unidad básica de almacenamiento; está formada por filas y columnas.
Vista	Representa lógicamente subjuegos de datos de una o varias tablas.
Secuencia	Generador de valor numérico.
índice	Mejora el rendimiento de algunas consultas.
Sinónimo	Proporciona nombres alternativos a objetos.

- Las tablas se pueden crear en cualquier momento, incluso mientras los usuarios utilizan la base de datos.
- No es necesario especificar el tamaño de las tablas, ya que se define en última instancia por la cantidad de espacio asignado a la base de datos entera. Sin embargo, es importante estimar cuánto espacio va a utilizar una tabla con el tiempo.

8.1.2. Reglas de Nomenclatura

Asignar un nombre a las columnas y tablas de base de datos siguiendo las reglas estándar para la nomenclatura de cualquier objeto de base de datos Oracle:

- Los nombres de tablas y columnas deben empezar por una letra y deben tener entre 1 y 30 caracteres.
- Los nombres solamente deben contener los caracteres A-Z, a-z, 0-9, _ (guión bajo), $ y # (caracteres válidos, aunque su uso no se recomienda).
- Los nombres no deben duplicar el nombre de otro objeto propiedad del mismo usuario de Oracle Server.
- Los nombres no deben ser palabras reservadas de Oracle Server.
- Instrucciones de Nomenclatura

Utilice nombres descriptivos para tablas y otros objetos de base de datos.

Nota: Los nombres no son sensibles a mayúsculas/minúsculas. Por ejemplo, EMPLOYEES se trata

Como el mismo nombre que eMPloyees o eMpLOYEES.

8.1.3. La Sentencia CREATE TABLE

Crear tablas para almacenar datos ejecutando la sentencia SQL CREATE TABLE, que es una de las sentencias del lenguaje de definición de datos (DDL), que se cubren en lecciones posteriores.

Las sentencias DDL son un subjuego de sentencias SQL utilizadas para crear, modificar o eliminar estructuras de base de datos Oracle10g.

Estas sentencias tienen un efecto inmediato sobre la base de datos y también registran información en el diccionario de datos.

Para crear una tabla, el usuario debe tener el privilegio CREATE TABLE y un área de almacenamiento en la que crear objetos. El administrador de la base de datos utiliza

secuencias del lenguaje de control de datos (DCL), que se cubren en una lección posterior, para otorgar privilegios a los usuarios.

```
CREATE TABLE [schema.]table(column data.type [DEFAULT expr]
[, ..] );
```

En la sintaxis:

schema	Es lo mismo que el nombre del propietario.
tabla	Es el nombre de la tabla.
DEFAULT exp.	Especifica un valor por defecto si se omite un valor en la sentencia INSERT.
column	Es el nombre de la columna
datatype	Es la longitud y el tipo de dato de la columna.

8.1.4. Referencia a Tablas de otro Usuario

Un esquema es una recopilación de objetos.

Los objetos de esquema son las estructuras lógicas que hacen referencia directamente a los datos de una base de datos.

Los objetos de esquema son tablas, vistas, sinónimos, secuencias, procedimientos almacenados, índices, agrupamientos y enlaces de base de datos.

Si una tabla no pertenece al usuario, el nombre del propietario debe ir como prefijo en la tabla. Por ejemplo, si hay un esquema llamado USER_B y USER_B tiene la tabla EMPLOYEES, debe especificar lo siguiente para recuperar datos de dicha tabla:

```
SELECT *
FROM   user_b.employees;
```

8.1.5. La Opción DEFAULT

A una columna se le puede asignar un valor por defecto utilizando la opción DEFAULT. Esta opción evita que se introduzcan valores nulos en las columnas si se inserta una fila sin un valor para la columna.

El valor por defecto puede ser un literal, una expresión o una función SQL, como SYSDATE y USER, pero el valor no puede ser el nombre de otra columna o una pseudo columna, como NEXTVAL o CURRVAL.

La expresión por defecto debe coincidir con el tipo de dato de la columna.

8.2. Creación de Tablas

En el ejemplo se crea la tabla DEPT, con tres columnas: DEPTNO, DNAME y LOC.

Además, se confirma la creación de la tabla emitiendo el comando DESCRIBE.

Como la creación de una tabla es una sentencia DDL, tiene lugar una validación automática cuando se ejecuta esta sentencia.

```
CREATE TABLE dep . (dep t no NUMBER ,
                    dname    VARCHAR2(13) ,
                    l oc     VARCHAR2(14))
```

Usar el comando DESCRIBE para confirmar la creación.

8.2.1. Tablas de la Base de Datos Oracle

Las tablas de usuario son tablas creadas por el usuario como, por ejemplo, EMPLOYEES. Hay otra recopilación de tablas y vistas en la base de datos Oracle conocida como el diccionario de datos.

Esta recopilación la crea y la mantiene Oracle Server y contiene información sobre la base de datos.

Todas las tablas del diccionario de datos son propiedad del usuario SYS.

El usuario no accede casi nunca a las tablas de base, ya que la información que contienen no es fácil de entender.

Por lo tanto, los usuarios normalmente acceden a vistas del diccionario de datos, donde la información se presenta en un formato más fácil de entender.

La información almacenada en el diccionario de datos incluye nombres de los usuarios de Oracle Server, privilegios otorgados a los usuarios, nombres de objetos de la base de datos, restricciones de tablas e información de auditoria.

Hay cuatro categorías de vistas del diccionario de datos; cada una tiene un prefijo distinto que refleja el uso que se pretende.

Prefijo	Descripción
USER_	Estas vistas contienen información sobre los objetos propiedad del usuario.
ALL_	Estas vistas contienen información sobre todas las tablas (tablas de objetos y tablas relaciónales) accesibles para el usuario.
DBA_	Estas vistas están restringidas y sólo pueden acceder a ellas personas que tienen asignado el rol DBA.
v$	Estas vistas son vistas de rendimiento dinámicas, rendimiento de servidor de base de datos, memoria y bloqueo.

8.2.2. Consulta del Diccionario de Datos

Puede consultar las tablas del diccionario de datos para visualizar los diversos objetos de base de datos que tenga en propiedad. Las tablas del diccionario de datos que se utilizan normalmente son:

- USER_TABLES
- USER_OBJECTS
- USER_CATALOG

Nota: USER_CATALOG tiene un sinónimo llamado CAT. Puede utilizar este sinónimo en lugar de USER_CATALOG en sentencias SQL.

```
SELECT      *
FROM              CAT;
```

8.2.3. Tipos de Dato

Tipo de dato	Descripción
VARCHAR2 (síze)	Dato de caracteres de longitud variable (se debe especificar un tamaño máximo: El tamaño mínimo es 1 y el máximo, 4000)

CHAR [(size)]	Datos de caracteres de longitud fija de bytes de tamaño de largo (el tamaño por defecto y mínimo es 1 ; el tamaño máximo es 2000)
NUMBER [(p, s)]	Número de precisión p y escala s (la precisión es el número total de dígitos decimales y la escala es el número de dígitos a la derecha de la coma décimal.
DATE	Valores de fecha y hora hasta el segundo más próximo entre el 1 de enero de 4712 a. C y el 31 de diciembre de 9999 d. C.
LONG	Dato de caracteres de longitud variable de hasta 2 GB
CLOB	Dato de caracteres de hasta 4 GB
RAW(size)	Dato raw binario de tamaño de longitud (se debe especificar un tamaño máximo. El tamaño máximo es 2000)
LONG RAW	Dato raw binario de longitud variable de hasta 2 GB
BLOB	Dato binario de hasta 4 GB
BFILE	Dato binario almacenado en un archivo externo; hasta 4 GB
ROWID	Sistema numérico de base 64 que representa la dirección única de una fila en su tabla

- No se copia una columna LONG cuando se crea una tabla utilizando una subconsulta.
- No se puede incluir una columna LONG en una cláusula GROUP BY u ORDER BY.
- Sólo se puede utilizar una columna LONG por tabla.
- No se pueden definir restricciones en una columna LONG.
- Puede utilizar una columna CLOB en lugar de una columna LONG

8.2.4. Formato ROWID

000000	**FFF**	**BBBBBB**	**RRR**
Número de objeto de datos	Número de archivo relativo	Número de bloque	Número de fila

Un ROWID ampliado se muestra con un esquema de codificación en base 64 que utiliza seis posiciones para el número de objeto de datos, tres posiciones para el número de archivo relativo, seis posiciones para el número de bloque y tres posiciones para el número de fila. El esquema de codificación en base 64 utiliza los caracteres A-Z, a-z, 0-9 y /. Esto constituye un total de 64 caracteres, tal y como se muestra en el siguiente ejemplo:

```
SQL> SELECT depa r tmen t_ id , r ow i d
2 FROM h r .depa r tmen ts ;
DEPARTMENT ID        ROWID
10           AAABQMAAFAAAAA6AAB
20           AAABQMAAFAAAAA6AAC
30           AAABQMAAFAAAAA6AAD
40           AAABQMAAFAAAAA6AAE
50           AAABQMAAFAAAAA6AAF
```

En este ejemplo:

- AAABQM es el número de objeto de datos.
- AAF es el número de archivo relativo.
- AAAAA6 es el número de bloque.
- AAA es el número de fila para el departamento con ID = 10

TambiÉn puedo usar el valor ROWID para ubicar un único registro

```
SELECT l as t _name , depa r tmen t _ id , job_ id , r ow i d
FROM EMPLOYEES
WHERE rowid    = 'AAAC9EAAEAAAABXAAU'
```

8.2.5. Creación de una tabla utilizando una subconsulta

Un segundo método para crear una tabla es aplicar la cláusula AS subquery, que crea la tabla e inserta filas devueltas desde la subconsulta.

```
CREATE   TABLE      tab le
[(co lumn ,      co lumn ... ) ] AS    subquery ;
```

En la sintaxis:

table es el nombre de la tabla.

column es el nombre de la columna, el valor por defecto y la restricción de integridad.

subquery es la sentencia SELECT que define el juego de filas que se van a insertar en la nueva tabla.

Instrucciones

- La tabla se crea con los nombres de columna especificados y las filas recuperadas por la sentencia SELECT se insertan en la tabla.
- La definición de columna sólo puede contener el nombre de columna y el valor por defecto.
- Si se proporcionan especificaciones de columna, el número de columnas debe ser igual al número de columnas de la lista SELECT de la subconsulta.
- Si no se proporcionan especificaciones de columna, los nombres de columna de la tabla son los mismos que los nombres de columna de la subconsulta.
- Las reglas de integridad no se transmiten a la nueva tabla, sólo las definiciones de tipo de dato de columna.

```
CREATE TABLE EMP_80 AS
SELECT employee_id, last_name, salary*12 ANNSAL, hire_date
FROM employees WHERE departmen t_ id =80
```

8.2.6. La Sentencia ALTER TABLE

Después de crear una tabla, es posible que tenga que cambiar la estructura de la misma por haber omitido una columna, porque sea necesario cambiar la definición de columna o porque tenga que eliminar columnas. Para ello, utilice la sentencia ALTER TABLE

```
ALTER TABLE t ab le
ADD (co lumn datatype [DEFAULT expr]
    [, co lumn datatype]... );
ALTER TABLE
MODIFY (column datatype [DEFAULT expr]
    [ , co l umn datatype]. ..);
ALTER TABLE t ab le DROP (co lumn ) ;
```

Puede agregar, modificar y borrar columnas de una tabla utilizando la sentencia ALTER TABLE.

En la sintaxis:

able : es el nombre de la tabla.

ADD|MODIFY|DROP: es el tipo de modificación

Column: es el nombre de la nueva columna.

Datatype: es el tipo de dato y la longitud de la nueva columna

DEFAULT expr: especifica el valor por defecto para una columna nueva

8.2.7. Adición de una columna

- Se puede agregar una nueva columna a una tabla
- No puede especificar dónde debe aparecer la columna.
- La nueva columna pasa a ser la última.

En el ejemplo se agrega una columna con el nombre JOB_ID a la tabla DEPTSO.

La columna JOB_ID pasa a ser la última de la tabla.

Nota: Si una tabla ya contiene filas cuando se agrega una columna, la nueva columna es inicialmente nula para todas las filas.

```
ALTER TABLE dep tSO
ADD                ( job_ id  VARCHAR2 (9) ) ;
Tab le a l t e r ad
```

Confirme con la sentencia DESCRIBE

8.2.8. Modificar una columna

Puede modificar una definición de columna utilizando la sentencia ALTER TABLE con la cláusula MODIFY.

La modificación de columna puede incluir cambios en el tipo de dato, el tamaño y el valor por defecto de la columna.

```
ALTER TABLE deptSO
MODIFY ( last_name VARCHAR2 (30 )} ;
```

- Puede aumentar el ancho o la precisión de una columna numérica.
- Puede aumentar el ancho de columnas numéricas o de caracteres.
- Puede disminuir el ancho de una columna sólo si ésta contiene sólo valores nulos o si la tabla no tiene filas.
- Puede cambiar el tipo de dato sólo si la columna contiene valores nulos.
- Puede convertir una columna CHAR en el tipo de dato VARCHAR2 o convertir una columna VARCHAR2 en el tipo de dato CHAR sólo si la columna contiene valores nulos o si no cambia el tamaño.
- Un cambio en el valor por defecto de una columna sólo afecta a las inserciones posteriores en la tabla.

8.2.9. Eliminar una columna

Puede borrar una columna de una tabla utilizando la sentencia ALTER TABLE con la cláusula DROP COLUMN. Esta función está disponible a partir de Oracle9i.

```
ALTER TABLE deptSO
DROP COLUMN job_id;
Table altered
```

- La columna puede o no puede contener datos.
- Con la sentencia ALTER TABLE, sólo se puede borrar una columna cada vez.
- La tabla debe tener al menos una columna después de la modificación.
- Una vez borrada una columna, no se puede recuperar

8.2.10. La opción SET UNSED

La opción SET UNUSED marca una o varias columnas como no utilizadas para que se puedan borrar cuando la demanda de los recursos del sistema sea menor.

Esta función está disponible a partir de Oracle9i.

Al especificar esta cláusula no se eliminan realmente las columnas de destino de cada fila de la tabla (es decir, no se restaura el espacio en disco utilizado por estas columnas).

Por lo tanto, el tiempo de respuesta es menor que si ejecutara la cláusula DROP.

Las columnas no utilizadas se consideran borradas, aunque sus datos permanezcan en las filas de la tabla.

Una vez marcada una columna como no utilizada, no tendrá acceso a ella.

Una consulta SELECT * no recuperará datos de columnas no utilizadas.

Además, los nombres y los tipos de las columnas marcadas como no utilizadas no se mostrarán durante un comando DESCRIBE y puede agregar una nueva columna a la tabla con el mismo nombre que la no utilizada.

```
La información de SET UNUSED se almacena en la vista de
diccionario USER_UNUSED_COL_TABS
ALTER TABLE table
SET UNUSED COLUMN column;
ALTER TABLE deptSO
SET UNUSED (last_name);
Table altered.
ALTER TABLE dep tSO
DROP UNUSED COLUMNS;
Table altered.
```

8.2.11. Cambio de nombre a un objeto

Las sentencias DDL adicionales incluyen la sentencia RENAME, que se utiliza para cambiar el nombre a una tabla, una vista, una secuencia o un sinónimo

```
RENAME dep t TO de ta i l_dep t;
Tab le renamed .
S in tax is
RENAME o l d_name TO new_name ;
```

old_name: es el nombre antiguo de la tabla, la vista, la secuencia o el sinónimo.

new_name: es el nombre nuevo de la tabla, la vista, la secuencia o el sinónimo.

Debe ser el propietario del objeto al que cambia el nombre.

8.2.12. Truncar una tabla

La sentencia TRÚNCATE TABLE es otra sentencia DDL que se utiliza para eliminar todas las filas de una tabla y para liberar el espacio de almacenamiento utilizado por dicha tabla. Al utilizar la sentencia TRÚNCATE TABLE, no puede realizar rollback de la eliminación de filas.

Sintaxis

```
TRUNCATE TABLE
```

Debe ser el propietario de la tabla o tener privilegios de sistema DELETE TABLE para truncar una tabla.

La sentencia DELETE también puede eliminar todas las filas de una tabla, pero no libera espacio de almacenamiento. El comando TRUNCATE es más rápido. Eliminar filas con la sentencia TRUNCATE es más rápido que con la sentencia DELETE por los siguientes motivos:

La sentencia TRUNCATE es una sentencia del lenguaje de definición de datos (DDL) y no genera información de rollback.

Truncar una tabla no arranca los disparadores de supresión de la tabla.

Si la tabla es la principal de una restricción de integridad referencial, no puede truncarla.

Desactive la restricción antes de emitir la sentencia TRUNCATE.

8.2.13. Agregar un comentario a una tabla

Puede agregar un comentario de hasta 2.000 bytes acerca de una columna, una tabla, una vista o una instantánea utilizando la sentencia COMMENT.

```
COMMENT ON TABLE emp loyees I S ' Emp l oyee I n f o r ma t ion ' ;
Commen t c rea t ed .
COMMENT ON TABLE t ab le \ COLUMN t ab le .co l umn
I S ' tex t ' ;
```

El comentario se almacena en el diccionario de datos y se puede visualizar en una de las siguientes vistas del diccionario de datos en la columna COMMENTS:

- ALL_COL_COMMENTS
- USER_COL_COMMENTS
- ALL_TAB_COMMENTS
- USER_TAB_COMMENTS

8.2.14. Truncamiento de una tabla

La sentencia TRUNCATE TABLE es otra sentencia DDL que se utiliza para eliminar todas las filas de una tabla y para liberar el espacio de almacenamiento utilizado por dicha tabla. Al utilizar la sentencia TRUNCATE TABLE, no puede realizar rollback de la eliminación de filas.

Sintaxis

TRUNCATE TABLE table;

En la sintaxis:

table es el nombre de la tabla

Debe ser el propietario de la tabla o tener privilegios de sistema DELETE TABLE para truncar una tabla.

La sentencia DELETE también puede eliminar todas las filas de una tabla, pero no libera espacio de almacenamiento. El comando TRUNCATE es más rápido. Eliminar filas con la sentencia TRUNCATE es más rápido que con la sentencia DELETE por los siguientes motivos:

La sentencia TRUNCATE es una sentencia del lenguaje de definición de datos (DDL) y no genera información de rollback.

Truncar una tabla no arranca los disparadores de supresión de la tabla.

Si la tabla es la principal de una restricción de integridad referencial, no puede truncarla. Desactive la restricción antes de emitir la sentencia TRUNCATE.

8.2.15. Eliminación de una tabla

La sentencia DROP TABLE elimina la definición de una tabla Oracle. Al borrar una tabla, la base de datos pierde todos los datos de la tabla y todos los índices asociados a ella.

Sintaxis

DROP TABLE table

En la sintaxis:

table es el nombre de la tabla.

Instrucciones

Todos los datos se suprimen de la tabla.

Las vistas y los sinónimos permanecen, pero no son válidos.

Se valida cualquier transacción pendiente.

Sólo pueden eliminar una tabla el creador de la misma o un usuario con el privilegio DROP ANY TABLE.

Nota: La sentencia DROP TABLE, una vez ejecutada, es irreversible. Oracle Server no cuestiona la acción cuando emite la sentencia DROP TABLE. Si es propietario de dicha tabla o tiene un privilegio de alto nivel, la tabla se elimina inmediatamente. Al igual que con todas las sentencias DDL, DROP TABLE se valida automáticamente.

8.2.16. Utilización de la cláusula PURGE

La base de datos Oracle 10g presenta una nueva función para borrar tablas. Al borrar una tabla, la base de datos no libera inmediatamente el espacio asociado a la tabla.

En su lugar, la base de datos cambia el nombre de la tabla y la coloca en una papelera de reciclaje, de donde se podrá recuperar después con la sentencia FLASHBACK TABLE si se da cuenta de que borró la tabla por error. Si desea liberar de forma inmediata el espacio asociado a la tabla en el momento de emitir la sentencia DROP TABLE, incluya la cláusula PURGE como se muestra en la sentencia de ejemplo.

Especifique PURGE sólo si desea borrar la tabla y liberar el espacio asociado a ella en un solo paso. Si especifica PURGE, la base de datos no coloca la tabla y sus objetos dependientes en la papelera de reciclaje.

La utilización de esta cláusula es equivalente a borrar primero la tabla y purgarla después de la papelera de reciclaje. Esta cláusula le ahorra un paso en el proceso. Proporciona también una seguridad mejorada si desea evitar que aparezca material sensible en la papelera de reciclaje.

Nota: No puede hacer rollback de una sentencia DROP TABLE con la cláusula PURGE, ni tampoco puede recuperar la tabla si la ha borrado con la cláusula PURGE. Esta función no estaba disponible en versiones anteriores.

DROP TABLE dept80 PURGE;

8.2.17. Sentencia FLASHBACK TABLE

Utilidad de Reparación de Autoservicio

La base de datos Oracle 10g proporciona un nuevo comando DDL de SQL, FLASHBACK

TABLE, para restaurar el estado de una tabla a un punto anterior en el tiempo en el caso de que la haya suprimido o modificado de forma accidental. El comando FLASHBACK TABLE es una herramienta de reparación de autoservicio para restaurar datos de una tabla junto con los atributos asociados como, por ejemplo, índices o vistas. Esto se consigue cuando la base de datos está online haciendo rollback sólo de los cambios posteriores en la tabla en cuestión.

Si se compara con mecanismos de recuperación tradicionales, esta función ofrece ventajas significativas, como la facilidad de uso, la disponibilidad y una recuperación más rápida.

También libera al DBA del trabajo de encontrar y restaurar propiedades específicas de aplicación. La función de flashback en tabla no se ocupa de la corrupción física provocada por

un disco en mal estado.

Sintaxis

```
FLASHBACK TABLE [schema . ] t ab l e [ , [ schema .]t ab le ] . . .
TO { T IMESTAMP | SCN } exp r
[ { ENABLE | D ISABLE } TR IGGERS ] ;
schema . t ab l e        Tab la a recupe ra r
T IMESTAMP |  SCN        Ind icado r ún ico / ho ra va l ida
```

Puede llamar a una operación de flashback en tabla en una o más tablas, incluso en tablas de diferentes esquemas. Para especificar el punto en el tiempo al que desea revertir, proporcione un registro de hora válido. Por defecto, los disparadores de base de datos están desactivados para todas las tablas implicadas. Para sustituir este comportamiento por defecto, especifique la cláusula ENABLE TRIGGERS.

Nota: Para obtener más información sobre la semántica de flashback y de papelera de reciclaje, consulte Oracle Database Administrator's Reference 10g Release 1 (10.1).

El ejemplo restaura la tabla EMP2 a un estado anterior a una sentencia DROP

Tabla Borrada

```
DROP TABLE emp2 ;
Tab le d r opped
```

Consulta para confirmar si se puede recuperar

```
SELECT o r ig ina l _name , ope r a t ion , d rop t ime ,
FROM recyc l eb in ;
```

Recuperación de la tabla

```
FLASHBACK TABLE emp2 TO BEFORE DROP ;
F lashback comp l e te
```

La papelera de reciclaje es en realidad una tabla de diccionario de datos que contiene información sobre objetos borrados. Las tablas borradas y los objetos asociados, como

índices, restricciones, tablas anidadas, etc., no se eliminan y siguen ocupando espacio. Siguen ocupando las cuotas de espacio de usuario, hasta que se purgan específicamente de la papelera de reciclaje o hasta que se produce una situación poco probable en la que las deba purgar la base de datos debido a restricciones de espacio de tablespace.

Se puede considerar a cada usuario propietario de una papelera de reciclaje, ya que, a menos que un usuario tenga el privilegio SYSDBA, los únicos objetos a los que puede acceder en la papelera de reciclaje son los de su propiedad.

Un usuario puede ver sus objetos en la papelera de reciclaje mediante la siguiente sentencia:

```
SELECT * FROM RECYCLEBIN;
```

Al borrar un usuario, los objetos que pertenecen a ese usuario no se colocarán en la papelera de reciclaje y se purgarán todos los objetos de la papelera de reciclaje.

Puede purgar la papelera de reciclaje con la siguiente sentencia:

```
PURGE RECYCLEBIN;
```

8.2.18. Tablas externas

Una tabla externa es una tabla de sólo lectura cuyos metadatos se almacenan en la base de datos pero cuyos datos se almacenan fuera de la base de datos. Esta definición de tabla externa se puede considerar una vista que se utiliza para ejecutar cualquier consulta SQL en datos externos sin necesidad de que se carguen primero los datos externos en la base de datos. Los datos de tabla externa se pueden consultar y unir directamente y en paralelo sin necesidad de que se carguen primero los datos externos en la base de datos. Puede utilizar SQL, PL/SQL y Java para consultar los datos en una tabla externa.

La diferencia principal entre las tablas externas y las normales es que las tablas organizadas externamente son de sólo lectura. No son posibles las operaciones DML y no se pueden crear índices en ellas. Sin embargo, se puede crear una tabla externa (y, por tanto, descargar datos) mediante el comando CREATE TABLE AS SELECT.

Oracle Server proporciona dos controladores de acceso principales para las tablas externas.

Uno de ellos es el controlador de acceso de cargador (u ORACLE_LOADER), que se utiliza para leer datos de archivos externos cuyo formato se puede interpretar mediante la utilidad

SQL*Loader. Observe que no se soporta toda la funcionalidad de SQL*Loader con tablas externas.

El controlador de acceso ORACLE_DATAPUMP se puede utilizar para importar y exportar datos mediante un formato independiente de la plataforma. El controlador de acceso

ORACLE_DATAPUMP escribe filas de una sentencia SELECT que se cargarán en una tabla externa como parte de la sentencia CREATE TABLE...ORGANIZATION

EXTERNAL...AS SELECT. Puede utilizar SELECT para leer datos de ese archivo de datos. También puede crear una definición de tabla externa en otro sistema y utilizar ese archivo de datos. Esto permite que se muevan datos entre bases de datos Oracle.

8.2.19. Creación de tablas externas

Utilice el comando CREATE DIRECTORY para crear un objeto de directorio. Un objeto de directorio especifica un alias para un directorio en el sistema de archivos del servidor en el que reside un origen de datos externo. Puede utilizar nombres de directorio al hacer referencia a un origen de datos externo, en lugar de predefinir el nombre de ruta de acceso de sistema operativo, para obtener una mayor flexibilidad en la gestión de archivos.

Debe tener privilegios del sistema CREATE ANY DIRECTORY para crear directorios. Al crear un directorio, se le otorgan automáticamente los privilegios de objeto READ y WRITE, y puede otorgar privilegios READ y WRITE a otros usuarios y roles. El DBA también puede otorgar estos privilegios a otros usuarios y roles.

Un usuario necesita privilegios READ para todos los directorios que se utilizan en las tablas externas a las que haya que acceder y privilegios WRITE para las ubicaciones de los archivos log, de errores y de desechos que se estén utilizando.

Además, es necesario un privilegio WRITE cuando el marco de tablas externas se utilice para descargar datos.

Oracle proporciona además el tipo ORACLE_DATAPUMP, con el que se pueden descargar datos (es decir, leer datos de una tabla de la base de datos e insertarlos en una tabla

externa) y recargarlos después en una base de datos Oracle. Se trata de una operación que sólo se realiza una vez y que se puede hacer al crear la tabla. Después de la creación y del relleno inicial, no se puede actualizar, insertar ni suprimir ninguna fila.

```
CREATE OR REPLACE DIRECTORY emp_dir AS '/.../emp_dir';
GRANT READ ON D I RECTORY emp_d i r TO h r ;
```

En la sintaxis:

OR REPLACE Especifique OR REPLACE para volver a crear el objeto de base de datos de directorio si ya existe. Puede utilizar esta cláusula para cambiar la definición de un directorio existente sin borrar, volver a crear y volver a otorgar privilegios de objeto de base de datos anteriormente otorgados en el directorio. Los usuarios a los que se otorgaron privilegios anteriormente en un directorio redefinido pueden seguir accediendo al directorio sin necesidad de que se vuelvan a otorgar los privilegios. Directory

Especifique el nombre del objeto de directorio que se va a crear. La longitud máxima del nombre de directorio es de 30 bytes. No se puede cualificar un objeto de directorio con un nombre de esquema.

'path_name' Especifique el nombre de ruta de acceso completa del directorio de sistema operativo en el resultado. Observe que el nombre de ruta de acceso es sensible a mayúsculas/minúsculas

Creación de una Tabla Externa

Cree tablas externas mediante la cláusula ORGANIZATION EXTERNAL de la sentencia

CREATE TABLE. De hecho, no está creando una tabla. En realidad, está creando metadatos en el diccionario de datos que se pueden utilizar para acceder a datos externos. Utilice la cláusula ORGANIZATION para especificar el orden en que se almacenarán las filas de datos de la tabla.

Al especificar EXTERNAL en la cláusula ORGANIZATION, indica que la tabla es de sólo lectura y que está ubicada fuera de la base de datos. Observe que los archivos externos deben existir ya fuera de la base de datos.

TYPE <access_driver_type> indica el controlador de acceso de la tabla externa. El controlador de acceso es la API (interfaz de programación de aplicaciones) que interpreta los datos externos para la base de datos. Si no especifica TYPE, Oracle utiliza el controlador de acceso por defecto, ORACLE_LOADER. La otra opción es ORACLE_DATAPUMP.

Utilice la cláusula DEFAULT DIRECTORY para especificar uno o más objetos de directorio de base de datos Oracle que se correspondan con directorios del sistema de archivos en los que puedan residir orígenes de datos externos.

La cláusula opcional ACCESS PARAMETERS le permite asignar valores a los parámetros del controlador de acceso específico para esta tabla externa.

```
CREATE TABLE < tab le_name>
( <co l_name> <da ta type> , … )
ORGAN IZAT ION EXTERNAL
( TYPE <access_d r ive r_ type>
DEFAULT D IRECTORY <d i r ec to ry_name> ACCESS PARAMETERS
(… ) )
LOCAT I ON ( '< loca t ion_spec i f ie r > ' ) )
REJECT L IM IT [ 0 | <numbe r> | UNL IM ITED ] ;
```

8.2.20. Consultas en tablas externas

Una tabla externa no describe ningún dato que esté almacenado en la base de datos. Tampoco describe cómo se almacenan los datos en el origen externo. En realidad, describe cómo debe presentar los datos al servidor el nivel de tabla externa. Corresponde al controlador de acceso y al nivel de tabla externa realizar las transformaciones necesarias en los datos del archivo de datos para que coincidan con la definición de tabla externa.

Cuando el servidor de bases de datos accede a datos de un origen externo, llama al controlador de acceso adecuado para obtener los datos de un origen externo en la forma que espera el servidor de bases de datos.

Es importante recordar que la descripción de los datos del origen de datos está separada de la definición de la tabla externa. El archivo de origen puede contener más o menos campos que el número de columnas de la tabla. Además, los tipos de datos del origen de datos pueden ser diferentes a las columnas de la tabla. El controlador de acceso se asegura de que los datos del origen de datos se procesen para que coincidan con la definición de la tabla externa.

```
SELECT      *
FROM o ldemp
```

8.2.21. Tablas Temporales

Además de las tablas normales ORACLE permite crear tablas temporales, que permite almacenar datos durante una transacción o sesión.

La sintaxis para crear una tabla temporal es

```
CREATE GLOBAL TEMPORARY TABLE table_name (col1,col2,....)
[ON COMM I T | PRESERVE ROWS ]
```

Los datos son accedidos desde la sesión que los llama únicamente y son propiedad exclusiva del owner.

Las tablas temporales pueden ser tratadas con sentencias DML como insert, update, delete y truncate.

Se pueden generar índices sobre las tablas temporales.

La persistencia de los datos va a depender de las cláusulas:

- ON COMMIT Preserva los datos durante una transacción
- PRESERVE ROWS Mantiene los datos durante toda la sesión

Las tablas temporales tienen almacenamiento físico y su definición se guarda en el diccionario de datos

Ejemplo

```
CREATE GLOBAL TEMPORARY TABLE tmp_emp ( empno NUMBER ( 5 ) ,
Empnom va rcha r2 (30 ) )
```

8.2.22. Tablas particionadas

Ante la necesidad de manejar grandes volúmenes de datos a los cuales acceder en forma rápida, es que ORACLE permite generar tablas particionadas.

A partir de la definición de una tabla se crean agrupamientos de datos lógicos que se implementan en forma física.

Si tomamos por ejemplo la tabla de ventas de una empresa, la misma puede ser particionada por rangos, por ejemplo el campo provincia por una condición lógica se puede crear una partición que responda a la siguiente condición

```
CREATE TABLE ven tas ( Venedo r Id (5 ) ,
                       VenNom va rcha r 2 ( 30 ) ,
                       VenEstaMonto number(10,2),
                       VenEsta   Varchar2(30))
PART I T IONED BY L IST (VenEs t a )
(PARTITION      norte VALUES ( 'SALTA', 'TUCUMAN'),
 PARTITION      centro VALUES('BUENOS AIRES','CORDOBA','SANTA
FE ' ),
 PARTITION sur VALUES ('CHUBUT','RIO NEGRO','NEUQUEN')
 PARTITION otros VALUES(DEFAULT));
```

Ejercicios:

1. Cree una tabla con el nombre emp_hist basada en la tabla employees
2. Agregue una columna antigüedad de tipo de dato numérico
3. Marque la columna creada como UNUSED y verifique la operación
4. Borre la columna marcada anteriormente
5. Truncar la tabla creada y verificar
6. Con el toad exportar a un archivo plano los datos de la tabla employees con el nombre de emp.txt
7. Crear una tabla externa con el nombre ext_emp y cargar los datos de la tabla emp.txt

9. Usuarios

9.1. Objetivos

En esta lección, aprenderá a controlar el acceso a base de datos para objetos específicos y a agregar nuevos usuarios con diferentes niveles de privilegios de acceso.

9.1.1. Control de Acceso de Usuarios

En un entorno de varios usuarios, necesita mantener la seguridad del acceso y el uso de la base de datos. Con la seguridad de base de datos de Oracle Server, puede:

- Controlar el acceso a la base de datos
- Otorgar acceso a objetos específicos de la base de datos
- Confirmar los privilegios otorgados y recibidos con el diccionario de datos Oracle
- Crear sinónimos para objetos de base de datos

La seguridad de base de datos se puede clasificar en dos categorías: seguridad del sistema y seguridad de los datos. La seguridad del sistema cubre el acceso y el uso de la base de datos en el nivel del sistema como, por ejemplo, nombre de usuario y contraseña, el espacio en disco asignado a los usuarios y las operaciones del sistema que pueden realizar los usuarios. La seguridad de datos cubre el acceso y el uso de los objetos de base de datos y las acciones que esos usuarios pueden llevar a cabo en los objetos.

9.1.2. Privilegios

Los privilegios son el derecho a ejecutar sentencias SQL en particular. El DBA (administrador de la base de datos) es un usuario de alto nivel con la capacidad de crear usuarios y de otorgarles acceso a la base de datos y a sus objetos. Los usuarios necesitan privilegios del sistema para obtener acceso a la base de datos y privilegios de

objeto para manipular el contenido de los objetos de la base de datos. A los usuarios también se les puede otorgar el privilegio de otorgar privilegios adicionales a otros usuarios o a roles, que son grupos especificados de privilegios relacionados.

Esquemas

Un esquema es una recopilación de objetos como, por ejemplo, tablas, vistas y secuencias. El esquema es propiedad de un usuario de base de datos y tiene el mismo nombre que el usuario.

Para obtener más información, consulte el manual de referencia Oracle Database 10g Application Developer's Guide - Fundamentals.

9.1.3. Privilegios del Sistema

Los usuarios y los roles tienen a su disposición más de 100 privilegios del sistema distintos.

Los privilegios del sistema suelen ser proporcionados por el administrador de la base de datos.

Privilegio del Sistema	Operaciones Autorizadas
CREATE USER	**La persona a la que se otorga el privilegio puede crear otros usuarios de Oracle**
DROP USER	**La persona a la que se otorga el privilegio puede borrar otro usuario.**
DROP ANY TABLE	**La persona a la que se otorga el privilegio puede borrar una tabla de cualquier esquema.**
BACKUP ANY TABLE	**La persona a la que se otorga el privilegio puede realizar copias de seguridad de cualquier esquema con la utilidad de exportación.**
SELECT ANY TABLE	**La persona a la que se otorga el privilegio puede consultar tablas, vistas o instantáneas en cualquier esquema.**
CREATE ANY TABLE	**La persona a la que se otorga el privilegio puede crear tablas en cualquier esquema.**

9.1.4. Creación de un Usuario

Para crear el usuario, el DBA ejecuta la sentencia CREATE USER.

El usuario no tiene ningún privilegio en ese momento.

Por tanto, el DBA puede otorgar privilegios a ese usuario.

Estos privilegios determinan lo que el usuario podrá hacer en el nivel de base de datos.

El ejemplo muestra la sintaxis resumida para crear un usuario.

```
CREATE USER use r IDENT IF IED BY pass
ACCOUNT UNLOCK ;
```

En la sintaxis:

user es el nombre del usuario que se va a crear

Password especifica que el usuario se debe conectar con esta contraseña

Accoun el estado operativo del usuario. Acepta como valores Lock/unlock

9.1.5. Privilegios de Usuario Típicos

Una vez creado el usuario, el DBA le puede asignar privilegios

Privilegio del Sistema	Operaciones Autorizadas
CREATE SESSION	**Conectarse a la base de datos**
CREATE TABLE	**Crear tablas en el esquema del usuario**
CREATE SEQUENCE	**Crear una secuencia en el esquema del usuario**
CREATE VIEW	**Crear una vista en el esquema del usuario**
CREATE PROCEDURE	**Crear un procedimiento, una función o un paquete en el esquema del usuario**

Sintaxis

```
GRANT privilege [, privilege...]
TO user [, user| role, PUBLIC...];
```

En la sintaxis

Privilge: es el privilegio del sistema que se va a otorgar

user |role|PUBLIC: es el nombre del usuario, el nombre del rol o, en el caso de PUBLIC, designa que el privilegio se otorga a todos los usuarios.

Los privilegios del sistema actuales se pueden encontrar en la vista de diccionario SESSION_PRIVS

9.1.6. Otorgamiento de Privilegios del Sistema

El DBA utiliza la sentencia GRANT para asignar privilegios del sistema al usuario. Una vez que se le han otorgado los privilegios al usuario, éste puede utilizarlos de forma inmediata.

En el ejemplo de la diapositiva, se han asignado al usuario Scott privilegios para crear sesiones, tablas, secuencias y vistas.

```
GRANT create session, create table,
create sequence, create view
TO scott;
Grant succeeded.
```

9.1.7. ¿Qué es un Rol?

Un rol es un grupo especificado de privilegios relacionados que se pueden otorgar al usuario. Este método facilita la revocación y el mantenimiento de privilegios.

Un usuario puede tener acceso a varios roles y se puede asignar a varios usuarios el mismo rol. Los roles se suelen crear para una aplicación de base de datos.

9.1.8. Creación y Asignación de un Rol

En primer lugar, el DBA debe crear el rol. Después, el DBA puede asignar privilegios al rol y asignar el rol a usuarios.

Sintaxis

CREATE ROLE role;

En la sintaxis:

role es el nombre del rol que se va a crear

Una vez creado el rol, el DBA puede utilizar la sentencia GRANT para asignar el rol a usuarios, del mismo modo que puede asignar privilegios al rol.

- Crear un rol

  CREATE ROLE manage r ;
  Ro l e c rea ted .

- Otorgar privilegios a un rol

  GRANT c r ea t e tab le , c rea t e v iew TO manage r ;
  G ran t succeeded .

- Otorgar un rol a usuarios

  GRANT manage r TO DE HAAN , KOCHHAR ;
  G ran t succeeded .

9.1.9. Cambio de Contraseñas

El DBA crea una cuenta e inicializa una contraseña para cada usuario. La contraseña se puede cambiar mediante la sentencia ALTER USER.

Sintaxis

ALTER USER user IDENTIFIED BY password;

En la sintaxis:

user es el nombre del usuario

password especifica la nueva contraseña

Aunque esta sentencia se puede utilizar para cambiar la contraseña, hay muchas otras opciones. Debe tener el privilegio ALTER USER para cambiar cualquier otra opción.

9.1.10. Privilegios de Objeto

Un privilegio de objeto es un privilegio o un derecho a realizar una acción determinada en una tabla, una vista, una secuencia o un procedimiento específicos. Cada objeto dispone de un juego determinado de privilegios que se pueden otorgar. La tabla de la diapositiva muestra los privilegios de varios objetos. Tenga en cuenta que los únicos privilegios que se aplican a una secuencia son SELECT y ALTER. UPDATE, REFERENCES e INSERT se pueden restringir mediante la especificación de un subjuego de columnas que se puedan actualizar.

Para restringir un privilegio SELECT, se puede crear una vista con un subjuego de columnas y otorgar el privilegio SELECT únicamente en la vista. Un privilegio otorgado en un sinónimo se convierte en un privilegio en la tabla base a la que haga referencia el sinónimo.

Privilegios de Objeto	Tabla	Vista	Secuencia	Procedimiento
ALTER	√		√	
DELETE	√	√		
EXECUTE				**√**
INDEX	√			
INSERT	√	√		
REFERENCES	√			
SELECT	√	√	√	
UPDATE	**√**	**√**		

9.1.11. Otorgamiento de Privilegios de Objeto

Existen diferentes privilegios de objeto disponibles para diferentes tipos de objetos de esquema.

Un usuario tiene automáticamente todos los privilegios de objeto para objetos de esquema contenidos en el esquema del usuario.

Un usuario puede otorgar cualquier privilegio de objeto en cualquier objeto de esquema que sea propiedad del usuario a cualquier otro usuario o rol.

Si el otorgamiento incluye WITH GRANT OPTION, la persona a la que se otorga el privilegio puede otorgar a su vez el privilegio de objeto a otros usuarios; de lo contrario, la persona a la que se otorga el privilegio lo puede utilizar pero no lo puede otorgar a otros usuarios.

En la sintaxis:

```
GRANT      object_priv [(columns)]
ON   object
TO   {user|role|PUBLIC}
     [WITH GRANT OPTION];
```

Sintaxis

object_priv	es un privilegio de objeto que se va a otorgar
ALL	especifica todos los privilegios de objeto
Columns	especifica la columna de una tabla o de una vista en la que se otorgan los privilegios
ON object	es el objeto en el que se otorgan privilegios
TO	identifica a quién se otorga el privilegio
PUBLIC	otorga privilegios de objeto a todos los usuarios
WITH GRANT OPTION	permite a la persona a la que se otorga el privilegio otorgar privilegios de objeto a otros usuarios y roles

9.1.12. Otorgamiento de Privilegios de Objeto

Instrucciones

- Para otorgar privilegios en un objeto, éste debe estar en el esquema o le deben haber otorgado los privilegios de objeto con la cláusula WITH GRANT OPTION.

- Un propietario de objeto puede otorgar cualquier privilegio de objeto a cualquier otro usuario o rol de la base de datos.

- El propietario de un objeto adquiere automáticamente todos los privilegios de objeto en ese objeto.

El primer ejemplo otorga a los usuarios Sue y Rich el privilegio para consultar la tabla EMPLOYEES.

```
GRANT select ON employees
TO sue, rich;
Grant succeeded.
```

El segundo ejemplo otorga privilegios UPDATE en columnas específicas de la tabla DEPARTMENTS a Scott y al rol de gestor.

```
GRANT update (department_name, location_id)
ON departments
TO scott, manager;
Grant succeeded.
```

Si Sue o Rich quieren utilizar ahora una sentencia SELECT para obtener datos de la tabla EMPLOYEES, la sintaxis que deben utilizar es:

```
SELECT * FROM HR.employees;
```

De forma alternativa, pueden crear un sinónimo para la tabla y emitir una sentencia SELECT desde el sinónimo:

```
CREATE SYNONYM emp FOR HR.employees;
SELECT * FROM emp;
```

9.1.13. Transferencia de Privilegios

La persona a la que se otorga un privilegio que se otorgue con la cláusula WITH GRANT OPTION lo puede transferir a otros usuarios y roles.

Los privilegios de objeto otorgados con la cláusula WITH GRANT OPTION se revocan si se revoca el privilegio del otorgante.

El ejemplo otorga al usuario Scott acceso a la tabla DEPARTMENTS con los privilegios para consultar la tabla y agregarle filas. El ejemplo muestra también que Scott puede otorgar a otros estos privilegios.

```
GRANT    select, insert ON departments
TO scott
WITH GRANT OPTION;
Grant succeeded.
```

Permita a todos los usuarios del sistema consultar datos de la tabla DEPARTMENTS de Alice.

```
GRANT select ON alice.departments
TO   PUBLIC;
Grant succeeded.
```

Palabra Clave PUBLIC

Un propietario de la tabla puede otorgar acceso a todos los usuarios mediante la palabra clave PUBLIC.

El segundo ejemplo permite a todos los usuarios del sistema consultar datos de la tabla DEPARTMENTS de Alice.

9.1.14. Confirmación de Privilegios Otorgados

Si intenta realizar una operación no autorizada, como suprimir una fila de una tabla para la que no tiene el privilegio DELETE, Oracle Server no permite que la operación se realice.

Si recibe el mensaje de error de Oracle Server "table or view does not exist", es porque ha realizado una de estas acciones:

- Ha especificado una tabla o una vista que no existen
- Ha intentado realizar una operación en una tabla o en una vista para la que no tiene el privilegio adecuado

Puede acceder al diccionario de datos para ver los privilegios de los que dispone.

El gráfico describe varias vistas de diccionario de datos

Vista del Diccionario de Datos	Descripción
ROLE_SYS_PRIVS	**Privilegios del sistema otorgados a roles**
ROLE_TAB_PRIVS	**Privilegios de la tabla otorgados a roles**
USER_ROLE_PRIVS	**Roles a los que puede acceder el usuario**
USER_TAB_PRIVS_MADE	**Privilegios de objeto otorgados en los objetos del usuario**
USER_TAB_PRIVS_RECD	**Privilegios de objeto otorgados al usuario**
USER_COL_PRIVS_MADE	**Privilegios de objeto otorgados en las columnas de los objetos del usuario**
USER_COL_PRIVS_RECD	**Privilegios de objeto otorgados al usuario en columnas específicas**
USER_SYS_PRIVS	**Privilegios del sistema otorgados al usuario**

9.1.15. Revocación de Privilegios de Objeto

Puede eliminar privilegios otorgados a otros usuarios mediante la sentencia REVOKE.

Al utilizar la sentencia REVOKE, los privilegios que especifique se revocarán de los usuarios que especifique y de cualquier otro usuario a quien el usuario revocado hubiera otorgado esos privilegios.

```
REVOKE {privilege [, privilege...]|ALL}
ON object
FROM {user[, user...]|role|PUBLIC}
[CASCADE CONSTRAINTS];
```

En la sintaxis:

CASCADE es necesario para eliminar cualquier restricción de integridad referencial realizada en el objeto CONSTRAINTS mediante el privilegio REFERENCES

Nota: Si revoca los privilegios de un usuario que debe dejar la compañía, debe volver a otorgar cualquier privilegio que este usuario hubiera otorgado a otros usuarios.

Si borra la cuenta de usuario sin revocarle los privilegios, esta acción no afectará a los privilegios del sistema otorgados por este usuario a otros usuarios.

```
REVOKE   select, insert
ON   departments
FROM        scott;
Revoke succeeded.
```

El ejemplo revoca los privilegios SELECT e INSERT otorgados al usuario Scott en la tabla DEPARTMENTS.

Nota: Si se otorga un privilegio a un usuario con la cláusula WITH GRANT OPTION, ese usuario también puede otorgar el privilegio con la cláusula WITH GRANT OPTION, de forma que es posible una larga cadena de personas a las que se otorgan privilegios, aunque no se permiten otorgamientos circulares. Si el propietario revoca un privilegio de un usuario que otorgó dicho privilegio a otros usuarios, se revocarán en cascada todos los privilegios otorgados.

Por ejemplo, si el usuario A otorga un privilegio SELECT en una tabla al usuario B con la cláusula WITH GRANT OPTION, el usuario B también puede otorgar al usuario C el

privilegio SELECT con la cláusula WITH GRANT OPTION y el usuario C puede otorgar al usuario D el privilegio SELECT. Si el usuario A revoca los privilegios del usuario B, los privilegios otorgados a los usuarios C y D también se revocan.

Ejercicios

1. Cree el usuario curso asignele una contraseña verifique si puede conectarse, sino realice las acciones necesarias para conectarse

2. Asigne los privilegios para que pueda acceder a las tablas employees y departments del esquema hr. Conéctese con el usuario cursos y verifique si realiza la consulta

3. Cree el rol cur_rol, otórguele los mismos privilegios del punto anterior

4. Elimine los privilegios al usuario curso

5. Cree el usuario curso1

6. Asigne permisos sobre el rol creado en el punto 3 a los usuarios curso y curso1 verifique el funcionamiento

7. Que tablas del diccionario de datos consultara para ver los privilegios de los usuarios curso y curos1

10. Objetos de la Base de datos

10.1. Objetivos

En este módulo aprenderá sobre los objetos que componen una base de datos, su uso, creación, modificación y eliminación de vistas, secuencias, sinónimos e índices.

10.1.1. Objetos de la base datos

Objeto	Descripción
Tabla	**Unidad básica de almacenamiento**
Vista	**Representa lógicamente subjuegos de datos de una o varias tablas**
Secuencia	**Genera valores numéricos**
Indice	**Mejora el rendimiento de algunas columnas**
Sinónimo	**Nombre alternativo de un objeto**

10.1.2. ¿Qué es una Vista?

Puede presentar subjuegos lógicos o combinaciones de datos mediante la creación de vistas de tablas.

Una vista es una tabla lógica basada en una tabla u otra vista.

Una vista no contiene datos propios, sino que es muy similar a una ventana a través de la cual se pueden visualizar o cambiar datos de tablas.

Las tablas en las que se basan las vistas se llaman tablas base. La vista se almacena como una sentencia SELECT en el diccionario de datos.

Las vistas restringen el acceso a los datos debido a que pueden mostrar columnas selectivas desde las tablas.

Las vistas se pueden utilizar para hacer que las consultas sencillas recuperen los resultados de consultas complicadas. Por ejemplo, las vistas se pueden utilizar para consultar información de varias tablas sin necesidad de que el usuario sepa cómo escribir una sentencia de unión.

Las vistas proporcionan independencia de datos para programas y usuarios ad-hoc. Una vista se puede utilizar para recuperar datos de diversas tablas.

Las vistas proporcionan a los grupos de usuarios acceso a los datos de acuerdo con sus criterios

10.1.3. Vistas Simples frente a Vistas Complejas

Hay dos clasificaciones para vistas: simples y complejas. La diferencia básica está relacionada con las operaciones DML (INSERT, UPDATE y DELETE).

Una vista simple es aquella que:

Deriva datos de sólo una tabla.

No contiene funciones ni grupos de datos.

Puede realizar operaciones DML a través de la vista.

Una vista compleja es aquella que:

Deriva datos de muchas tablas.

Contiene funciones o grupos de datos.

No siempre permite operaciones DML a través de la vista.

10.1.4. Creación de una Vista

Puede crear una vista si embebe una subconsulta dentro de la sentencia CREATE VIEW

```
CRÉATE [OR REPLACE] [FORCÉ |NOFORCE ] V IEW v iew
[ (a l i as ! , a l i as ] . . . ) ] AS subque ry
[WITH CHECK OPTION [CONSTRAINT constraint] ]
[WITH READ ONLY [CONSTRAINT cons traint]];
```

En la sintáxis:

OR REPLACE	**vuelve a crear la vista si ya existe**
FORCÉ	**crea la vista independientemente de si las tablas base existen o no**
NOFORCE	**crea la vista sólo si las tablas base existen (es el valor por defecto).**
view	**Nombre de la vista**
alias	**Especifica nombres para las columnas** **(El número de alias debe coincidir con el número de expresiones seleccionadas en la vista.)**
subquery	**es una sentencia SELECT completa (puede utilizar alias para las columnas de la lista SELECT).**
WITH CHECK OPTION	**especifica que sólo las filas accesibles a la vista se pueden insertar o actualizar**
constraint WITH	**es el nombre asignado a la restricción CHECK OPTION.**
READ ONLY	**asegura que no se pueda realizar ninguna operación DML en esta vista.**

10.1.5. Creación de una Vista

Instrucciones para la creación de una vista:

La subconsulta que define una vista puede contener sintaxis SELECT compleja, incluyendo uniones, grupos y subconsultas.

La subconsulta que define la vista no puede contener una cláusula ORDER BY. La cláusula ORDER BY se especifica cuando se recuperan datos de la vista.

Sino especifica un nombre de restricción para una vista creada con WITH CHECK OPTION, el sistema asigna un nombre por defecto con el formato SYS_Cn.

Puede utilizar la opción OR REPLACE para cambiar la definición de la vista sin borrarla y volver a crearla o volver a otorgar privilegios de objeto previamente otorgados.

Sintaxis para crear una vista que muestra a los empleados del departamento 80, esta vista toma el nombre de columnas de los utilizados en el subquery

```
CREATE VIEW empvuSO AS
SELECT    employee_id, last_name, salary
  FROM      employees
  WHERE    department_id =80;
  View created.
```

Otra alternativa consiste en utilizar un alias después de la sentencia CREATE y antes de la subconsulta SELECT. El número de alias enumerado debe coincidir con el número de expresiones seleccionadas en la subconsulta.

```
CREATE VIEW      salvuSO ( ID_NUMBER / NAME ,ANN_SALARY ) AS SELECT
employee_id/ last_name, salary*12
FROM      employees
WHERE    department_id = 50;
View created .
```

10.1.6. Recuperación de Datos de una Vista

Puede recuperar datos de una vista de la misma forma que los recupera de una tabla. Puede visualizar los contenidos de toda la vista o sólo de columnas y filas específicas.

```
SELECT * FROM empvuSO
```

10.1.7. Información sobre las vistas

Una vez que se haya creado la vista, puede consultar la vista del diccionario de datos llamada USER_VIEWS para ver el nombre y la definición de la vista.

El texto de la sentencia SELECT que constituye la vista se almacena en una columna LONG.

Acceso a Datos Utilizando Vistas

Cuando accede a datos utilizando una vista, Oracle Server realiza las siguientes operaciones:

1. Recupera la definición de la vista de la tabla USER_VIEWS del diccionario de datos.

2. Comprueba los privilegios de acceso para la tabla base de la vista.

3. Convierte la consulta de la vista en una operación equivalente en las tablas o tabla base subyacente. Dicho de otro modo, los datos se recuperan de las tablas base o se realiza una actualización de ellas.

10.1.8. Modificación de una Vista

Con la opción OR REPLACE, se puede crear una vista incluso si ya existe otra con el mismo nombre, sustituyendo así la versión antigua de la vista para su propietario. Esto significa que la vista se puede modificar sin necesidad de borrar, volver a crear u otorgar, de nuevo, privilegios de objeto.

Nota: Cuando asigne alias de columna en la cláusula CREATE VIEW, recuerde que los alias se enumeran en el mismo orden que las columnas de la subconsulta.

10.1.9. Creación de una Vista Compleja

En el ejemplo se crea una vista compleja de nombres de departamento, salarios mínimos, salarios máximos y salarios medios por departamento.

Observe que se han especificado nombres alternativos para la vista.

Esto es un requisito si alguna columna de la vista se deriva de una función o una expresión.

Puede visualizar la estructura de la vista utilizando el comando DESCRIBE de /SQL*Plus. Visualice los contenidos de la vista emitiendo una sentencia SELECT.

```
CREATE V I EW dep t_suxn_vu ( name , m insa l , maxsa l , avgsa l )
AS SELECT d .depa r txnen t_name , M IN (e .sa la r y ) ,  MAX (e .sa la ry )
 , AVG (e . sa la ry )
FROM emp l oyees e , depa r tmen t s d
```

```
WHERE e.department_id = d.department_id
GROUP BY d.department_name;
View created.
```

10.1.10. Realización de Operaciones DML en una Vista

Puede realizar operaciones DML en datos a través de una vista si dichas operaciones siguen ciertas reglas.

No puede eliminar una fila de una vista que contenga:

- Funciones de grupo
- Una cláusula GROUP BY
- La palabra clave DISTINCT
- La palabra clave ROWNUM de pseudocolumna

No se puede agregar datos a traves de una vista si existe alguna de las condiciones indicadas anteriormente o haya columnas NOT NULL sin valores por defecto en la tabla base, que no esten seleccionadas en la vista

10.1.11. Uso de la Cláusula WITH CHECK OPTION

Es posible realizar comprobaciones de integridad referencial a través de las vistas.

También puede forzar las restricciones en el nivel de la base de datos.

La vista se puede utilizar para proteger la integridad de datos, pero el uso está muy limitado.

La cláusula WITH CHECK OPTION especifica que las inserciones (INSERT) y las actualizaciones (UPDATE) realizadas a través de la vista no pueden crear filas que la vista no pueda seleccionar y, por tanto, permite que las restricciones de integridad y las comprobaciones de validación de datos se fuercen en los datos que se insertan o se actualizan.

Si hay un intento de realizar operaciones DML en filas que la vista no haya seleccionado, se muestra un mensaje de error junto con el nombre de la restricción, si se ha especificado.

10.1.12. Denegación de Operaciones DML

Puede asegurarse de que no se produce ninguna operación DML en la vista si la crea con la opción WITH READ ONLY. En el ejemplo de la transparencia se modifica la vista EMPVU10 para evitar cualquier operación DML en la vista.

```
CREATE OR REPLACE V IEW empvu lO
( emp l oyee_numbe r , emp loyee_name , job_ t i t l e ) AS SELECT
emp loyee_ id , l as t _name , job_ i d
FROM        emp loyees
WHERE       depa r tmen t_ id = 10
  W ITH READ ONLY ;
```

Cualquier intento de eliminar una fila de una vista con una restricción de sólo lectura da como resultado un error.

```
DELETE FROM  empvu lO
WHERE emploYee_number =200
DELETE FROM empvu lO
ERROR a t l ine 1 :
ORA -01752 : canno t  de le te f rom  v iew  w i thou t  exac t l y  one  key -
p rese rved  t ab le
```

Cualquier intento de insertar o modificar una fila utilizando la vista con una restricción de sólo lectura da como resultado un error de Oracle Serven

01733: virtual column not allowed here.

10.14 Eliminación de una Vista

Puede utilizar la sentencia DROP VIEW para eliminar una vista. Esta sentecia elimina la definición de a vista de la base de datos. La eliminación de vistas no tiene ningún efecto en las tablas en las que se basaba la vista. Las vistas u otras aplicaciones basadas en vistas suprimidas se invalidan. Solamente pueden eliminar una vista el creador o un usuario con el privilegio DROP ANY VIEW.

```
DROP VIEW empSO;
```

10.1.13. Vistas en Línea

Una vista en línea se crea colocando una subconsulta en la cláusula FROM y asignando un alias a la subconsulta. Ésta define un origen de datos al que se puede hacer referencia en la consulta principal. En el siguiente ejemplo, la vista en línea b devuelve los detalles de todos los números de departamento y el salario máximo de cada departamento de la tabla EMPLOYEES. La cláusula WHERE a. department_id = b.department_id AND a.salary < b.maxsal de la consulta principal muestra nombres de empleado, salarios, números de departamento y salarios máximos para todos los empleados que ganan menos que el salario máximo de su departamento

```
SELECT a. last_name, a.salary, a.department_id, b.maxsal
FROM   employees a, (SELECT department_id, max(salary)
    maxsal
FROM   employees
                GROUP BY department_id) b
WHERE  a.department_id = b.department_id
AND a.salary < b.maxsal;
```

10.16 Análisis de los "N Principales"

Las consultas de N principales son útiles en supuestos en los que es necesario mostrar solamente los n registros principales o los n menos importantes de una tabla basada en una condición. Este juego de resultados se puede utilizar para otros análisis. Por ejemplo, utilizando análisis de los N principales puede realizar los siguientes tipos de consulta:

- Los tres empleados de la compañía que más ganan
- Las cuatro contrataciones más recientes de la compañía
- Los dos representantes de ventas que han vendido el máximo número de productos

Los tres productos que han tenido las máximas ventas en los últimos seis meses

Las consultas de N principales utilizan una estructura de consulta anidada consistente con los elementos que se describen a continuación:

Una subconsulta o una vista en línea para generar la lista ordenada de datos. La subconsulta o la vista en línea incluyen la cláusula ORDER BY para asegurar que la clasificación está en el orden deseado. Para que los resultados recuperen los valores más grandes es necesario un parámetro DESC.

Una consulta externa para limitar el número de filas en el juego de resultados final. La consulta externa incluye los siguientes componentes:

La pseudo columna ROWNUM, que asigna un valor secuencial que comienza por 1 a cada una de las filas devuelta en la subconsulta.

Una cláusula WHERE, que especifica las n filas que se van a devolver. La cláusula externa WHERE debe utilizar un operador < o <=.

```
SELECT      [column_list], ROWNUM ¨
FROM (SELECT      [column_list]
FROM table
ORDER    BY Top-N_column)
      WHERE    ROWNUM <=    N;
```

10.1.14. Práctica 1

1. Cree una vista llamada EMPLOYEES_VU basada en los números y los nombres deempleado y los números de departamento de la tabla EMPLOYEES. Cambie la cabecera para el nombre de empleado a EMPLOYEE.

2. Visualice los contenidos de la vista EMPLOYEES VU.

3. Seleccione el nombre de vista y el texto de la vista del diccionario de datos USER_VIEWS. Nota: Ya existe otra vista. EMP_DETAILS_VIEW se creó como parte del esquema.

10.1.15. ¿Qué es una Secuencia?

Una secuencia es un objeto de base de datos creado por un usuario que se puede compartir con varios usuarios para generar enteros únicos.

Un uso típico de las secuencias es la creación de un valor de clave primaria, que debe ser único para cada fila.

Una rutina interna de Oracle genera y aumenta (o disminuye) la secuencia. Esto puede ser un objeto que ahorre tiempo ya que puede reducir la cantidad de código de aplicación necesario para escribir una rutina generadora de secuencias.

Los números de secuencia se almacenan y se generan independientemente de las tablas. Por lo tanto, la misma secuencia se puede utilizar para varias tablas.

10.1.16. Creación de una Secuencia

```
CREATE SEQUENCE sequence
[ INCREMENT BY n ]
[ START W I TH n ]
[{MAXVALUE n \ NOMAXVALUE}]
[{MINVALUE n \ NOMINVALÜE}]
[ {CYCLE | NOCYCLE } ]
[ {CACHE n | NOCACHE } ] ;
```

En la sintaxis

Sequence	**es el nombre del generador de secuencias.**
INCREMENT BY n	**especifica el intervalo entre números de secuencia donde n es un entero (si esta cláusula se omite, la secuencia aumenta en 1).**
START WITH n	**especifica el primer número de secuencia que se va a generar (si esta cláusula se omite, la secuencia comienza por 1).**
MAXVALUE n	**especifica el valor máximo que la secuencia puede generar.**
NOMAXVALUE	**especifica un valor máximo de 10^A27 para una secuencia ascendente y -1 para una secuencia descendente (ésta es la opción por defecto).**
MINVALUE n	**especifica el valor de secuencia mínimo.**
NOMINVALUE	**especifica un valor mínimo de 1 para una secuencia ascendente y (10^A26) para una secuencia descendente (ésta es la opción por defecto).**
CYCLE | NOCYCLE	**especifica si la secuencia continua generando valores después de alcanzar su valor máximo o mínimo (NOCYCLE es la opción por defecto).**
CACHE n | NOCACHE	**especifica cuántos valores preasigna Oracle Server y cuántos mantiene en memoria (por defecto,**

Oracle Server almacena en caché 20 valores).

```
CREATE SEQUENCE dept_deptid_seq
    I NCREMENT BY 10
    START W ITH
    MAXVALUE 9999
    NOCACHE NOCYCLE ;
Sequence c rea t ed .
```

10.1.17. Confirmación de Secuencias

Una vez que haya creado la secuencia, ésta se documenta en el diccionario de datos.

Puesto que una secuencia es un objeto de base de datos, puede identificarla en la tabla USER_OBJECTS del diccionario de datos.

También puede confirmar la definición de la secuencia seleccionando desde la vista del diccionario de datos USER_SEQUENCES.

10.1.18. Pseudo columnas NEXTVAL y CURRVAL

Después de crear la secuencia, ésta genera números secuenciales para el uso en tablas. Haga referencia a los valores de secuencia utilizando las pseudo columnas NEXTVAL y CURRVAL.

La pseudo columna NEXTVAL se utiliza para extraer números de secuencia sucesivos de una secuencia especificada. Debe cualificar NEXTVAL con el nombre de la secuencia. Al hacer referencia a seguence. NEXTVAL, se genera un nuevo número de secuencia y el número actual se coloca en CURRVAL.

La pseudo columna CURRVAL se utiliza para hacer referencia a un número de secuencia que el usuario actual acaba de generar. NEXTVAL se debe utilizar para generar un número de secuencia en la sesión del usuario actual antes de que se pueda hacer referencia a CURRVAL. Debe cualificar CURRVAL con el nombre de la secuencia. Al hacer referencia a seguence. CURRVAL, se muestra el último valor devuelto al proceso de dicho usuario.

10.1.19. Uso de una Secuencia

```
INSERT INTO departments(department_id,
department_name, location_id)
VALÚES          (dept_deptid_seq.NEXTVAL, 'Support', 2500);
1 row created.
```

En el ejemplo se inserta un departamento nuevo en la tabla DEPARTMENTS.

Se utiliza la secuencia DEPT_DEPTID_SEQ para generar un nuevo número de departamento como se indica a continuación: Puede visualizar el valor actual de la secuencia:

SELECT dept_deptid_seq.CURRVAL FROM dual;

Suponga ahora que desea contratar a empleados para el nuevo departamento.

La sentencia INSERT que va a ejecutar para todos los empleados nuevos puede incluir el siguiente código:

```
INSERT INTO employees (employee_id/ department_id, ...)
VALUES (employees_seq.NEXTVAL/ dept_deptid_seq .CURRVAL,
...);
```

Nota: En el ejemplo anterior se asume que ya se ha creado una secuencia llamada EMPLOYEE_SEQ para generar nuevos números de empleado.

Almacene en memoria caché las secuencias para proporcionar un acceso más rápido a dichos valores de secuencia.

El caché se rellena la primera vez que se hace referencia a la secuencia. Cada solicitud para el siguiente valor de secuencia se recupera desde la secuencia almacenada en caché. Después de que se haya utilizado el último valor de secuencia, la siguiente solicitud introduce otro caché de secuencias en la memoria.

Intervalos en la Secuencia

Aunque los generadores de secuencias emiten números secuenciales sin intervalos, esta acción se produce independientemente de una validación o rollback. Por lo tanto, si realiza rollback en una sentencia que contenga una secuencia, se pierde el número.

Otro evento que puede producir intervalos en la secuencia es un fallo del sistema. Si la secuencia almacena en memoria caché valores, estos se pierden si el sistema falla.

Como las secuencias no están unidas a tablas directamente, la misma secuencia se puede utilizar para varias tablas. De este modo, cada tabla puede contener intervalos en los números secuenciales.

10.1.20. Modificación de una Secuencia

Si alcanza el límite MAXVALUE para la secuencia, no se asignará ningún valor adicional de la secuencia y recibirá un mensaje de error que indicando que la secuencia excede el límite MAXVALUE. Para continuar utilizando la secuencia, puede modificarla utilizando la sentencia ALTER SEQUENCE.

Sintaxis

```
ALTER SEQUENCE
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUEJ}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Instrucciones para la Modificación de Secuencias

Debe ser el propietario o tener el privilegio ALTER para la secuencia para modificarla.

Sólo se ven afectados los números de secuencia futuros por la sentencia ALTER SEQUENCE.

La opción START WITH no se puede cambiar utilizando ALTER SEQUENCE. La secuencia se debe borrar y volver a crear para reiniciarla en un número diferente.

Se realiza alguna validación. Por ejemplo, no se puede imponer un nuevo MAXVALUE que sea menor que el número de secuencia actual.

```
ALTER SEQUENCE dept_deptid_seq
INCREMENT BY 20
MAXVALUE 90
NOCACHE
NOCYCLE ;
```

```
ALTER SEQUENCE dep t_dep t i d_seq *
ERROR a t l ine 1 :
ORA -04009 : MAXVALUE canno t be made to be l ess than the
current va l ué
```

10.1.21. Eliminación de una Secuencia

Para eliminar una secuencia del diccionario de datos, utilice la sentencia DROP SEQUENCE. Debe ser el propietario de la secuencia o tener el privilegio DROP ANY SEQUENCE para eliminarla.

Sintaxis

```
DROP  SEQUENCE  sequence ;
```

10.1.22. ¿Qué son los índices?

Un índice de Oracle Server es un objeto de esquema que puede acelerar la recuperación de filas utilizando un puntero. Los índices se pueden crear explícitamente o automáticamente. Si no tiene un índice en la columna, se realiza una exploración completa de la tabla.

Un índice proporciona acceso directo y rápido a las filas de una tabla. Su objetivo es reducir la necesidad de E/S de disco utilizando una ruta indexada para encontrar los datos rápidamente. Oracle Server utiliza y mantiene el índice automáticamente. Una vez que se ha creado el índice, no se requiere ninguna actividad directa por parte del usuario.

Los índices son independientes lógica y físicamente de su tabla indexada. Esto significa que se pueden crear o borrar en cualquier momento y que no tienen efecto en las tablas base ni en otros índices.

Nota: Al borrar una tabla, los índices correspondientes también se borran.

10.1.23. Clasificación de índices

Un índice es una estructura de árbol que permite el acceso directo a una fíla de una tabla.

Los índices se pueden clasificar en función de su disefío lógico o de su implementación física. En la clasificación lógica, los índices se agrupan desde el punto de vista de la aplicación, mientras que en la clasificación física, por el modo en que se almacenan índices concatenados y de columna única

Un índice de columna única tiene una única columna en la clave de índice; por ejemplo, un índice en la columna de números employees de la tabla employees.

Un índice concatenado, también conocido como índice compuesto, se crea en múltiples columnas de una tabla. Las columnas de un índice concatenado no tienen que estar en el mismo orden que las columnas de la tabla ni tampoco ser adyacentes; por ejemplo, un índice basado en las columnas de los departamentos y los puestos de trabajo de una tabla de empleados.

El número máximo de columnas de un índice de clave compuesta es 32. Sin embargo, el tamaño combinado de todas las columnas no puede exceder, aproximadamente, la mitad (menos algo de sobrecarga) del espacio de datos disponible de un bloque de datos índices únicos y no únicos

Los índices pueden ser únicos o no únicos. Los índices únicos garantizan que no haya dos filas de una tabla con valores duplicados en la columna (o columnas) clave. Los índices no únicos no imponen esta restricción sobre los valores de columna índices basados en funciones

Un índice basado en función se crea cuando se utilizan funciones o expresiones que implican a una o más columnas de la tabla que se está indexando. Un índice basado en función calcula previamente el valor de la función o la expresión y lo almacena en el índice. Los índices basados en funciones se pueden crear como un índice B-Tree o como un índice de bitmaps índices de dominios

Un índice de dominio es un índice específico de una aplicación (Text, Spatial) que se crea, gestiona y accede mediante rutinas proporcionadas por un tipo de índice. Se denomina índice de dominio porque indexa los datos de dominios específicos de la aplicación.

Sólo se soportan índices de dominio de una columna. Se pueden generar índices de dominio de columna única en columnas con tipos de dato escalares, de objeto o LOB. Índices particionados y no particionados

Los índices particionados se utilizan para que las tablas de gran tamaño almacenen entradas de índice correspondientes a un índice en varios segmentos. El particionamiento permite distribuir un índice entre muchos tablespaces, lo que disminuye la contención para búsquedas de índice y aumenta la capacidad de gestión. Los índices particionados se suelen utilizar con tablas particionadas para mejorar la escalabilidad y la capacidad de gestión. Se puede crear una partición de índice para cada partición de tabla.

En esta lección se explica la creación y el mantenimiento de los índices no particionados B-

Tree y de bitmap.

10.1.24. Indice B-Tree

Aunque todos los índices utilizan una estructura B-Tree, el término índice B-Tree se suele asociar a un índice que almacena una lista de ROWID para cada clave.

Estructura de un índice B-Tree

En la parte superior del índice está la raíz, que contiene entradas que apuntan al siguiente nivel del índice. En el siguiente nivel están los bloques de derivación, que a su vez apuntan a los bloques del siguiente nivel del índice. En el nivel inferior se encuentran los nodos hoja, que contienen las entradas de índice que apuntan a las filas de la tabla. Los bloques hoja están doblemente enlazados para facilitar la exploración del índice en orden ascendente y descendente de valores clave.

Formato de las entradas de hoja de índice

Una entrada de índice incluye los siguientes componentes:

- Una cabecera de entrada, que almacena el número de columnas y la información de bloqueo
- Los pares valor-longitud de la columna clave, que definen el tamaño de una columna en la clave seguida del valor de la columna (el número de dichos pares es el número máximo de columnas del índice)
- ROWID de una fila, que contiene los valores clave de entradas de hoja de índice

En un índice B-Tree de una tabla no particionada:

- Los valores clave se repiten si hay múltiples filas que tienen el mismo valor clave, a menos que el índice esté comprimido.
- No hay ninguna entrada de índice correspondiente a una fila que tenga todas las columnas clave NULL. Por lo tanto, una cláusula WHERE que especifique NULL siempre dará lugar a una exploración completa de la tabla.
- El ROWID restringido se utiliza para apuntar a las filas de la tabla, ya que todas las filas pertenecen al mismo segmento.

Efecto de las operaciones DML en un índice

Oracle Server mantiene todos los índices cuando se realizan operaciones DML en la tabla. He aquí una explicación del efecto de un comando DML en un índice:

- Las operaciones de inserción dan como resultado la inserción de una entrada de índice en el bloque apropiado.
- La supresión de una fila sólo provoca la supresión lógica de la entrada de índice. El espacio que utiliza la fila suprimida no esta disponible para las nuevas entradas hasta que se suprimen todas las entradas del bloque.
- Las actualizaciones de las columnas clave conducen a una supresión lógica y a una inserción en el índice.

10.1.25. Indices de Bitmap

Los índices de bitmap tienen más ventajas que los índices B-Tree en determinadas situaciones:

- Si una tabla tiene millones de filas y las columnas clave tienen una cardinalidad baja; es decir, hay muy pocos valores distintos para la columna. Por ejemplo, los índices de bitmap son preferibles a los índices B-Tree para las columnas de sexo y estado civil de una tabla que contiene registros de pasaportes.
- Cuando las consultas suelen utilizar una combinación de múltiples condiciones WHERE con relación al operador OR.
- Si hay poca actividad de actualización o de sólo lectura en las columnas clave

10.1.26. Comparación entre los índices B-Tree y Bitmap

Los índices de bitmap son más compactos que los índices B-Tree cuando se utilizan con columnas de baja cardinalidad.

Las actualizaciones de columnas clave de un índice de bitmap son más costosas porque los bitmaps utilizan bloqueos a nivel del segmento de bitmap, mientras que en un índice B-Tree los bloqueos se realizan en entradas correspondientes a filas individuales de la tabla.

Los índices de bitmap se pueden utilizar para realizar operaciones como, por ejemplo, una operación booleana con bitmaps. Oracle Server puede utilizar dos segmentos de bitmap para realizar una operación booleana en bits y obtener un bitmap resultante. Esto permite utilizar eficazmente los bitmaps en las consultas que usan el predicado booleano.

En resumen, los índices B-Tree pueden ser más adecuados para indexar tablas dinámicas en un entorno OLTP, mientras que los índices de bitmap pueden ser útiles en entornos de almacenamiento de datos en los que se realizan consultas complejas en tablas de gran tamaño y estáticas.

10.1.27. ¿Cómo Se Crean los índices?

Se pueden crear dos tipos de índices. Uno de ellos es un índice único: Oracle Server lo crea automáticamente al definir una columna en una tabla para obtener una restricción PRIMARY KEY o de clave UNIQUE. El nombre del índice es el nombre que se ha asignado a la restricción.

El otro tipo de índice es un índice no único, que puede crear un usuario. Por ejemplo, puede crear un índice de columna FOREIGN KEY para una unión en una consulta para mejorar la velocidad de recuperación.

Nota: Puede crear manualmente un índice único, pero se recomienda que cree una restricción única, lo cual crea implícitamente un índice único.

10.1.28. Creación de un índice B-TREE

Cree un índice en una o varias columnas emitiendo la sentencia CRÉATE INDEX.

```
CREATE I NDEX index
ON t ab l a ( co lumn , co lumn1 ] . . . ) ;
```

En la sintaxis:

índex	es el nombre del índice.
table	es el nombre de la tabla.
column	es el nombre de la columna en la tabla que se va a indexar.

10.1.29. Creación de índices de Bitmap Sintaxis

Utilice el siguiente comando para crear un índice de bitmap

```
CREATE B I TMAP I NDEX í nd ice
ON  t ab l a  ( co lumna      ,      co lumna1?
```

Tenga en cuanta que un índice de bitmap no puede ser único.

10.1.30. Cuando crear índices

Más índices en una tabla no significa consultas más rápidas. Cada operación DML que se valida en una tabla con índices significa que los índices se deben actualizar. Cuantos más índices tenga asociados a una tabla, más esfuerzo tiene que realizar Oracle Server para actualizar todos los índices después de una operación DML.

Cuándo Se Crea un índice

Por lo tanto, debe crear índices sólo si:

- La columna contiene un amplio rango de valores.
- La columna contiene un gran número de valores nulos.
- Una o más columnas se utilizan juntas frecuentemente en una cláusula WHERE o en una condición de unión.
- La tabla es grande y se espera que la mayoría de las consultas recuperen menos del 2-4 % de las filas.

Recuerde que si desea forzar la unicidad, debe definir una restricción única en la definición de tabla. A continuación, se crea automáticamente un índice único.

10.1.31. Cuando no crear un índice

Normalmente no merece la pena crear un índice si:

- La tabla es pequeña.

- Las columnas no se suelen utilizar como condición en la consulta.
- • Se espera que la mayoría de las consultas recuperen más del 2-4 por ciento de las filas de la tabla.
- La tabla se actualiza frecuentemente.
- Se hace referencia a las columnas indexadas como parte de una expresión.

10.1.32. Confirmación de índices

Confirme la existencia de los índices desde la vista USER_INDEXES del diccionario de datos.

También puede verificar las columnas afectadas por un índice consultando la vista

USER_ IND_COLUMNS .

10.1.33. Índices Basados en Funciones

Los índices basados en funciones definidos con las palabras clave UPPER (col umn_name) o LOWER (column_name) permiten búsquedas no sensibles a mayúsculas/minúsculas. Por ejemplo, el siguiente índice:

```
CREATE   INDEX upper_last_name_idx ON employees
(UPPER(last_name));
```

facilita el procesamiento de consultas como:

```
SELECT * FROM emp loyees WHERE UPPER ( las t_name ) = 'K ING ' ;
```

Para asegurarse de que Oracle Server utiliza el índice en lugar de realizar una exploración completa de la tabla, compruebe que el valor de la función no es nulo en consultas posteriores. Por ejemplo, se garantiza que la siguiente sentencia utiliza el índice, pero sin la cláusula WHERE es posible que Oracle Server realice una exploración completa de la tabla:

```
SELECT              *
```

```
FROM        emp loyees
WHERE       UPPER ( l as t _name ) I S NOT NULL
ORDER BY UPPER ( las t_name ) ;
```

Oracle Server trata los índices con columnas marcadas DESC como índices basados en funciones. Las columnas marcadas DESC se ordenan en orden descendente.

10.1.34. Eliminación de un índice

Los índices no se pueden modificar. Para cambiar un índice, debe borrarlo y volver a crearlo. Elimine una definición de índice del diccionario de datos emitiendo la sentencia DROP INDEX. Para borrar un índice, debe ser el propietario del mismo o tener el privilegio DROP ANY INDEX.

DROP INDEX index;

En la sintaxis:

índex es el nombre del índice.

Nota: Si borra una tabla, los índices y las restricciones se borran automáticamente, pero las vistas y las secuencias se mantienen.

10.1.35. Identificación de índices no Utilizados

A partir de Oracle9i, se pueden recopilar estadísticas acerca del uso de un índice y se pueden mostrar en la vista V$OBJECT_USAGE.

Si la información recopilada indica que un índice no se utiliza nunca, ese índice se puede borrar. Asimismo, al eliminar los índices no utilizados se reduce la sobrecarga que requiere Oracle Server para las operaciones DML, lo que mejora el rendimiento. Cada vez que se especifica la cláusula MONITORING USAGE, se restablece V$OBJECT_USAGE para el índice especificado.

La información anterior se limpia o se restablece, y se registra la nueva hora de inicio.

ALTER INDEX index

MONITORING USAGE;Activa el monitoreo del indice

ALTER INDEX index

NOMONITORING USAGE; Desctiva el monitoreo del indice

SELECT * FROM V$OBJECT_USAGE
Co lumnas de la v i s t a V$OBJECT_USAGE

INDEX_NAME: Nombre del índice

TABLE_NAME: La tabla correspondiente

MONITORING: Indica si el control está definido en ON u OFF

USED: Devuelve YES o NO, si se ha utilizado el índice durante el tiempo de control

START_MONITORING: Hora de inicio del control del índice

END_MONITORING: Hora de parada del control del índice

10.1.36. Obtención de Información acerca de los índices

Para obtener información acerca de los índices, consulte las siguientes vistas:

• DBA_INDEXES: Proporciona información acerca de los índices

• DBA_IND_COLUMNS: Proporciona información acerca de las columnas indexadas

• V$OBJECT_USAGE: Proporciona información acerca del uso de un índice

10.1.37. Creación y eliminación de un Sinónimo

Para consultar una tabla propiedad de otro usuario, es necesario que escriba en el nombre de la tabla el nombre del usuario que la creó como prefijo, seguido de un punto.

Al crear un sinónimo se elimina la necesidad de cualificar el nombre de objeto con el esquema y se proporciona el nombre alternativo para una tabla, vista, secuencia, procedimiento u otros objetos.

Este método puede ser especialmente útil con nombres largos de objeto, como por

ejemplo, las vistas.

```
CREATE [ PUBL I C ] SYNONYM synonym
FOR      ob j ec t ;
```

En la sintaxis:

PUBLIC	crea un sinónimo accesible para todos los usuarios.
synonym	es el nombre del sinónimo que se va a crear.
object	identifica el objeto para el que se crea el sinónimo.

Instrucciones

- No se puede contener el objeto en un paquete.
- El nombre de un sinónimo privado debe ser distinto de todos los demás objetos propiedad del mismo usuario.

Para borrar un sinónimo, utilice la sentencia DROP SYNONYM. Solamente el administrador de la base de datos puede borrar un sinónimo público.

DROP PUBLIC SYNONYM dept; Synonym dropped.

10.2. Ejercicios

1. Cree una secuencia para utilizarla con la columna de clave primaria de la tabla DEPT La secuencia debe comenzar en 200 y tener un valor máximo de 1000. Haga que la secuencia aumente en 10 números. Asigne a la secuencia el nombre DEPT_ID_SEQ
2. Inserte 2 registros en la tabla dept asegurandose de utilizar la secuencia creada para la columna dept_id.
3. Visualice los índices y la unicidad que existe en el diccionario de datos para la tabla employees.
4. Utilize la correspondiente para monitorear el uso de los indices de la tabla employees, para las columnas department_id , last_name y first_name

11. Operadores Set y funciones Avanzadas

11.1. Objetivo

En esta lección aprendera a manejar los operadores set y realizar uniones, uniones excluyentes, intersecciones de tablas.

11.1.1. Operadores SET

Los operadores SET combinan los resultados de consultas de dos o más componentes en un resultado. Las consultas que contienen operadores SET se denominan consultas compuestas.

Operador	Devuelve
UNION	**Todas las filas distintas seleccionadas por cualquiera de las dos consultas**
UNION ALL	**Todas las filas seleccionadas por cualquiera de las dos consultas, incluidos todos los duplicados**
INTERSECT	**Todas las filas distintas seleccionadas por ambas consultas**
MINUS	**Todas las filas seleccionadas por la primera sentencia SELECT y no seleccionadas en la segunda sentencia SELECT**

Todos los operadores SET tienen la misma prioridad. Si una sentencia SQL contiene varios operadores SET, Oracle Server las evaluará de izquierda (arriba) a derecha (abajo) si no hay paréntesis que especifiquen explícitamente otro orden. Debe utilizar paréntesis para especificar explícitamente el orden de evaluación en consultas que utilicen el operador

INTERSECT con otros operadores SET.

11.1.2. Operador UNION

El operador UNION devuelve todas las filas seleccionadas por cualquiera de las dos consultas. Utilice el operador UNION para recuperar las filas seleccionadas de las distintas tablas sin duplicados

Instrucciones

- El número de columnas y los tipos de datos de las columnas que se están seleccionando deben ser idénticos en todas las sentencias SELECT utilizadas en la consulta. No es necesario que los nombres de las columnas sean idénticos.
- UNION opera sobre todas las columnas que se están seleccionando.
- Los valores NULL no se ignoran durante la comprobación de duplicados.
- El operador IN tiene una prioridad más alta que el operador UNION.
- Por defecto, la salida se clasifica por orden ascendente de la primera columna de la cláusula SELECT.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

11.1.3. Operador UNION ALL

Utilice el operador UNION ALL para devolver todas las filas de varias consultas. Instrucciones

Las instrucciones para UNION y UNION ALL son las mismas, con estas dos excepciones relacionadas con UNION ALL:

A diferencia de lo que sucede con UNION, las filas duplicadas no se eliminan y la salida no se ordena por defecto.

No se puede utilizar la palabra clave DISTINCT.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY employee_id;
```

11.1.4. Operador INTERSECT

Utilice el operador INTERSECT para devolver las filas que son comunes a varias consultas. Instrucciones

- El número de columnas y los tipos de datos de las columnas que se están seleccionando mediante las sentencias SELECT deben ser idénticos en todas las sentencias SELECT utilizadas en la consulta. No es necesario que los nombres de las columnas sean idénticos.
- Invertir el orden de las tablas interseccionadadas no altera el resultado.
- INTERSECT no ignora los valores NULL.

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

11.1.5. Operador MINUS

Utilice el operador MINUS para devolver las filas devueltas por la primera consulta que no estén presentes en la segunda (la primera sentencia SELECT menos (MINUS) la segunda sentencia SELECT).

Instrucciones

- El número de columnas y los tipos de datos de las columnas que se están seleccionando mediante las sentencias SELECT deben ser idénticos en todas las

sentencias SELECT utilizadas en la consulta. No es necesario que los nombres de las columnas sean idénticos.

- Para que funcione el operador MINUS, todas las columnas de la cláusula WHERE deben estar en la cláusula SELECT.

```
SELECT employee_id,job_id
FROM   employees
MINUS
SELECT employee_id,job_id
FROM   job_history;
```

11.1.6. Instrucciones para los Operadores SET

Las expresiones de las listas SELECT deben corresponder en número y en tipo de datos.

Las consultas que utilizan operadores UNION, UNION ALL, INTERSECT y MINUS en su cláusula WHERE deben tener el mismo número y el mismo tipo de columnas de su lista SELECT.

La cláusula ORDER BY:

Sólo puede aparecer al final de la sentencia

Aceptará el nombre de columna, un alias o la notación posicional

El nombre de columna, o el alias, si se utiliza en una cláusula ORDER BY, debe ser de la primera lista SELECT.

Los operadores SET se pueden utilizar en subconsultas.

Cuando una consulta utiliza operadores SET, Oracle Server elimina automáticamente las filas duplicadas excepto en el caso del operador UNION ALL.

Los nombres de columna de la salida los decide la lista de columnas de la primera sentencia SELECT. Por defecto, la salida se clasifica por orden ascendente de la primera columna de la cláusula SELECT.

Las expresiones correspondientes de las listas SELECT de las consultas componentes de una consulta compuesta deben corresponder en número y en tipo de datos.

SI las consultas componentes seleccionan datos de carácter, el tipo de datos de los valores de retorno se determinan de este modo:

Si ambas consultas seleccionan valores el tipo de datos CHAR, los valores devueltos tienen el tipo de datos CHAR.

Si alguna de las dos o las dos consultas seleccionan valores del tipo de datos

VARCHAR2, los valores devueltos tienen el tipo de datos VARCHAR2.

11.1.7. Correspondencia de Sentencias SELECT

Como las expresiones de las listas SELECT de las consultas deben corresponder en número, puede utilizar columnas ficticias y las funciones de conversión de tipo de datos para cumplir esta regla. En la diapositiva, se proporciona el nombre location como cabecera de columna ficticia. Se utiliza la función TO_NUMBER de la primera consulta para hacerla corresponder con el tipo de datos NUMBER de la columna LOCATION_ID recuperada por la segunda consulta. De forma parecida, se utiliza la función TO_DATE de la segunda consulta para hacerla corresponder con el tipo de datos NUMBER de la columna LOCATION_ID recuperada por la primera consulta.

```
SELECT department_id, TO_NUMBER(null),location, hire_date
FROM employees
UNION
SELECT department_id, location_id,TO_DATE(null)
FROM departments;
```

11.1.8. Control del Orden de Filas

Por defecto, la salida se clasifica por orden ascendente de la primera columna.

Puede utilizar la cláusula ORDER BY para cambiar esto.

La cláusula ORDER BY sólo se puede utilizar una vez en una consulta compuesta.

Si se utiliza, la cláusula ORDER BY se debe colocar al final de la consulta.

La cláusula ORDER BY acepta el nombre de columna o un alias. Sin la cláusula ORDER BY, el código de ejemplo de la diapositiva genera la siguiente salida por orden alfabético de la primera columna:

Nota: Piense en una consulta compuesta en la que se utilice el operador SET UNION más de una vez. En este caso, la cláusula ORDER BY sólo puede utilizar posiciones y no expresiones explícitas.

```
SELECT 'O rac le ' AS " My d ream " , 3 a_dummy
FROM dua l
UN I ON
SELECT ' S is temas i n teg rados ' , 1 a_dummy
FROM dua l
UN I ON
SELECT ' I T ' , 2 a_dummy
FROM dua l
ORDER BY a_dummy ;
```

11.2. Practica

1. El departamento de recursos humanos necesita una lista de identificadores de departamento que no contengan el identificador de puesto ST_CLERK. Utilice los operadores SET para crear este informe.

2. El departamento de recursos humanos necesita una lista de países en los que no haya ningún departamento. Muestre el identificador de país y el nombre de los países.

3. Utilice los operadores SET para crear este informe.

4. Genere una lista de puestos para los departamentos 10, 50 y 20, en ese orden. Muestre el identificador de puesto y el de departamento mediante operadores SET.

12. Recuperación Jerárquica Objetivos

Al finalizar esta lección, debería estar capacitado para interpretar el concepto de consulta jerárquica, crear un informe en estructura de árbol formatear datos jerárquicos, excluir ramas de la estructura de árbol

12.1. Concepto de consultas jerárquicas

Mediante consultas jerárquicas, puede recuperar datos basándose en una relación jerárquica natural entre filas de una tabla. Una base de datos relacional no almacena registros de forma jerárquica. Sin embargo, si existe una relación jerárquica entre las filas de una sola tabla, un proceso denominado desplazamiento por el árbol permite que se construya la jerarquía. Una consulta jerárquica es un método de informar de las ramas de un árbol en un orden específico.

Imagine un árbol genealógico con los miembros mayores de la familia situados cerca de la base o del tronco del árbol y los más jóvenes representados como ramas del árbol. Las ramas pueden tener más ramas, y así sucesivamente.

Nota: Los árboles jerárquicos se utilizan en diversos campos como, por ejemplo, la genealogía humana (árboles genealógicos), la cría de ganado (reproducción), la gerencia empresarial (jerarquías de los supervisores), la manufactura (ensamblaje de productos), la investigación evolutiva (desarrollo de especies) y la investigación científica.

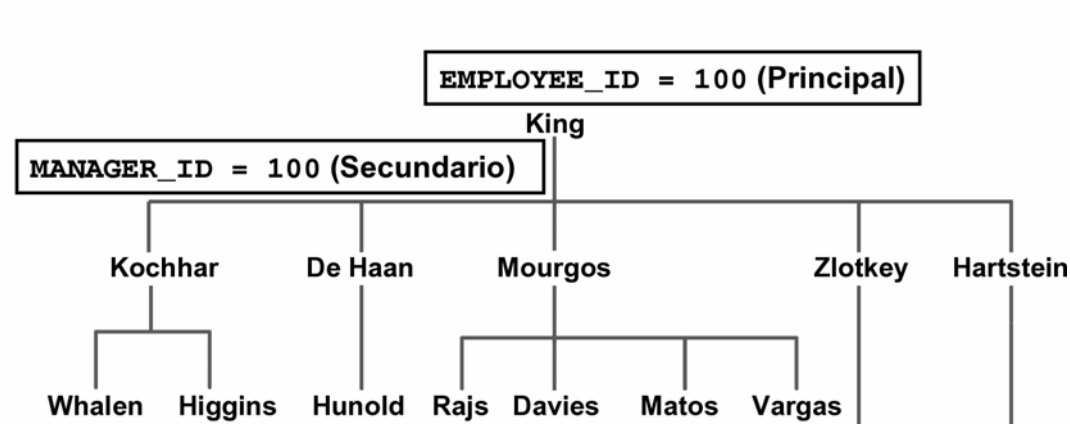
12.1.1. Estructura de Árbol Natural

La tabla EMPLOYEES tiene una estructura de árbol que representa la línea de supervisores.

Para crear la jerarquía, se puede observar la relación entre valores equivalentes de las columnas EMPLOYEE_ID y MANAGER_ID. Para utilizar esta relación, se puede unir la tabla a sí misma. La columna MANAGER_ID contiene el número de empleado del supervisor del empleado.

La relación principal-secundaria de una estructura de árbol le permite controlar:

- La dirección de desplazamiento por la jerarquía
- El punto de partida dentro de la jerarquía



12.1.2. Consultas jerárquicas

Las consultas jerárquicas se pueden identificar por la presencia de las cláusulas CONNECT BY y START WITH.

```
SELECT [ LEVEL ] , co lumn , exp r . . .
FROM       t abl e
[ WHERE cond i t ion (s ) ]
[ START W I TH cond i t ion (s ) ]
[CONNECT BY PR IOR cond i t ion (s ) ] ;
```

En la sintaxis:

SELECT	Es la cláusula SELECT estándar
LEVEL	Para cada fila devuelta por una consulta jerárquica, la pseudocolumna LEVEL devuelve 1 para una fila de raíz, 2 para un secundario de la raíz y así sucesivamente.
FROM table	Especifica la tabla, la vista o la instantánea que contiene las columnas. Puede seleccionar de una sola tabla.
WHERE	Restringe las filas devueltas por la consulta sin que afecte a otras filas de

	la jerarquía.
condition	**Es una comparación con expresiones**
START WITH	**Especifica las filas de raíz de la jerarquía (dónde comenzar). Esta cláusula es necesaria para una consulta jerárquica verdadera.**
CONNECT BY PRIOR	**Especifica las columnas en las que existe la relación entre filas principales y secundarias** **Esta cláusula es necesaria para una consulta jerárquica**

La sentencia SELECT no puede contener una unión ni una consulta de una vista que contenga una unión.

12.1.3. Desplazamiento por el Árbol

La cláusula START WITH determina la fila o las filas que se deben utilizar como raíz del árbol. La cláusula START WITH se puede utilizar en conjunción con cualquier condición válida.

Ejemplos

Mediante la tabla EMPLOYEES, parta de King, el presidente de la compañía.

```
START W ITH manage r _ id l S NULL
```

Mediante la tabla EMPLOYEES, parta del empleado Kochhar. Una condición START WITH puede contener una subconsulta.

```
... START WITH emp loyee_i d = (SELECT emp l oyee_ id
    FROM          emp l oyees
    WHERE      l as t_name = ' Kochha r ')
```

Si se omite la cláusula START WITH, el desplazamiento por el árbol parte de todas las filas de la tabla como filas de raíz. Si se utiliza una cláusula WHERE, el desplazamiento parte de todas las filas que satisfacen la condición WHERE. Esto ya no refleja una jerarquía verdadera.

Nota: Las cláusulas CONNECT BY PRIOR y START WITH no son del estándar ANSI SQL.

La dirección de la consulta, ya sea de principal a secundario o de secundario a principal, está determinada por la colocación de la columna CONNECT BY PRIOR.

El operador PRIOR se refiere a la fila principal. Para encontrar las filas secundarias de una fila principal, Oracle Server evalúa la expresión PRIOR para la fila principal y las demás expresiones para cada fila de la tabla. Las filas para las que la condición es verdadera son las filas secundarias de la principal. Oracle Server selecciona siempre las filas secundarias evaluando la condición CONNECT BY con respecto a una fila principal actual.

Ejemplos

Desplácese de arriba abajo mediante la tabla EMPLOYEES. Defina una relación jerárquica en la que el valor EMPLOYEE_ID de la fila principal sea igual al valor MANAGER_ID de la fila secundaria.

```
.. . CONNECT BY PR IOR emp l oyee_ id = manage r _ id
Desp l ácese de aba jo a r r iba med ian te la tab l a EMPLOYEES .
.. . CONNECT BY PR IOR manage r _ id = emp loyee_ id
```

No es necesario codificar inmediatamente el operador PRIOR después de CONNECT BY. Así, la siguiente cláusula CONNECT BY PRIOR genera el mismo resultado que la del ejemplo anterior.

```
CONNECT BY emp loyee_ id = PR IOR manage r_ i d
```

Nota: La cláusula CONNECT BY no puede contener una subconsulta.

12.1.4. Desplazamiento por el Árbol: De Abajo Arriba

En el ejemplo se muestra una lista de supervisores que comienza por el empleado cuyo identificador es 101.

Ejemplo

En el ejemplo siguiente, los valores de EMPLOYEE_ID se evalúan para la fila principal y MANAGER_ID, y los valores de SALARY son evaluados por las filas secundarias. El operador PRIOR se aplica sólo al valor de EMPLOYEE_ID.

```
.. . CONNECT BY PR IOR emp l oyee_ id = manage r _ id
```

```
AND sa l a ry > 15000 ;
```

Para calificarse como fila secundaria, una fila debe tener un valor de MANAGER_ID igual al valor EMPLOYEE_ID de la fila principal y debe tener un valor SALARY superior a 15.000 pesos

```
SELECT employee_id, last_name, job_id, manager_id
FROM   employees
START WITH   employee_id = 101
CONNECT BY PRIOR manager_id = employee_id;
```

12.1.5. Desplazamiento por el Árbol: De Arriba Abajo

Desplazándose de arriba abajo, muestre los nombres de los empleados y su supervisor. Utilice el empleado King como punto de partida. Imprima sólo una columna.

```
SELECT   last_name || ' reports to ' ||
PRIOR last_name "Walk Top Down"
FROM   employees
START WITH last_name = 'King'
CONNECT BY PRIOR employee_id = manager_id;
```

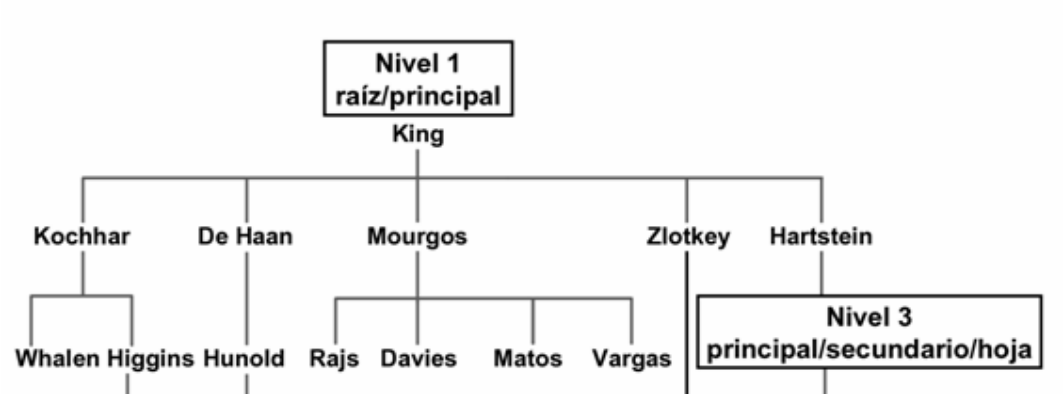
12.1.6. Clasificación de Filas con la Pseudocolumna LEVEL

Puede mostrar explícitamente la clasificación o el nivel de una fila en la jerarquía mediante la pseudo columna LEVEL.

Esto hará que el informe resulte más legible. Las bifurcaciones en las que salen una o varias ramas de una rama mayor se denominan nodos y el extremo de una rama se denomina hoja, o nodo hoja. El diagrama de la diapositiva muestra los nodos del árbol invertido con sus valores LEVEL. Por ejemplo, el empleado Higgins es principal y secundario, mientras que el empleado Davies es secundario y hoja.

Pseudo columna LEVEL

Valor	Nivel
1	**Nodo raíz**
2	**Secundario de un nodo raíz**
3	**Secundario de un secundario y así sucesivamente.**



12.1.7. Formato de Informes Jerárquicos mediante LEVEL

A los nodos de un árbol se les asignan números de niveles desde la raíz.

Utilice la función LPAD en conjunción con la pseudo columna LEVEL para mostrar un informe jerárquico como árbol con sangrado.

En el ejemplo

LPAD(char1,n [,char2]) devuelve char1, rellenado a la izquierda hasta la longitud n con la secuencia de caracteres en char2. El argumento n es la longitud total del valor de retorno

LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')define el formato de visualización.

char1 es LAST_NAME, n la longitud total del valor de retorno total es la longitud de

LAST_NAME +(LEVEL*2)-2 y char2 es '_'.

Dicho de otro modo, esto indica a SQL que tome LAST_NAME y lo rellene a la izquierda con el carácter '_' hasta que la longitud de la cadena resultante sea igual a los valores determinados por LENGTH(last_name)+(LEVEL*2)-2.

Para King, LEVEL = 1. Por tanto, (2 * 1) - 2 = 2 - 2 = 0. Así pues, King no se rellena con ningún carácter '_' y se muestra en la columna 1.

Para Kochhar, LEVEL = 2. Por tanto, (2 * 2) - 2 = 4 - 2 = 2 . Así pues, Kochhar se rellena con 2 caracteres '_' y se muestra con sangrado.

El resto de los registros de la tabla EMPLOYEES se muestra de forma parecida.

```
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2,'_')
AS org_chart
FROM employees
START WITH last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

12.1.8. Eliminación de Ramas

Puede utilizar las cláusulas WHERE y CONNECT BY para eliminar el árbol; es decir, para controlar qué nodos o qué filas se muestran. El predicado que utilice funciona como condición booleana.

Ejemplos

Partiendo de la raíz, desplácese de arriba abajo y elimine el empleado Higgins del resultado, pero procese las filas secundarias.

```
SELECT    department_id, employee_id, last_name, job_id, salary
FROM      employees
WHERE     last_name  != 'Higgins'
START     WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

Partiendo de la raíz, desplácese de arriba abajo y elimine el empleado Higgins del resultado y todas las filas secundarias.

```
SELECT    department_id, employee_id, last_name, job_id, salary
FROM      employees
START     WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
```

```
AND las t_name != ' H i gg i ns ' ;
```

12.2. Práctica

1. Genere un informe que muestre un organigrama del departamento de Mourgos. Imprima los apellidos, los salarios y los identificadores de departamento.

2. Cree un informe que muestre la jerarquía de los supervisores del empleado Lorentz. Muestre primero el supervisor inmediato.

13. Expresiones Regulares

13.1. Objetivos

En esta lección, aprenderá a utilizar la función de soporte de expresiones normales que se ha introducido en la base de datos Oracle 10g.

13.2. Visión General de Expresiones Regulares

La base de datos Oracle 10g introduce el soporte de expresiones regulares.

La implementación cumple con el estándar POSIX (Sistema Operativo Portátil para UNIX), controlado por el IEEE (Instituto de Ingenieros en Electricidad y Electrónica), para la semántica y la sintaxis de correspondencia de datos ASCII. Las capacidades multilingües de Oracle amplían las capacidades de correspondencia de los operadores más allá del estándar POSIX. Las expresiones regulares son un método para describir patrones sencillos y complejos de búsqueda y manipulación.

La manipulación y la búsqueda de cadenas suponen un amplio porcentaje de la lógica de una aplicación basada en la Web.

El uso va desde la simple búsqueda de las palabras "San Francisco" en un texto especificado, pasando por la compleja extracción de todas las direcciones URL del texto, hasta la búsqueda más compleja de todas las palabras cuyo segundo carácter sea una vocal.

Si se une al SQL nativo, el uso de expresiones regulares permite operaciones muy potentes de búsqueda y de manipulación de cualquier dato almacenado en una base de datos Oracle.

Puede utilizar esta función para solucionar fácilmente problemas que de otro modo resultarían muy complejos de programar.

13.2.1. Meta caracteres

Los meta caracteres son caracteres especiales que tienen un significado especial como, por ejemplo, un comodín, un carácter de repetición, un carácter de no correspondencia o un rango de caracteres. Puede utilizar varios símbolos de metacaracteres predefinidos en la correspondencia de patrones.

Símbolo	Descripción
*	**Se corresponde con cero o más incidencias**
|	**Operador de modificación para especificar correspondencias alternativas**
^/$	**Se corresponde con el inicio de línea/fin de línea**
[ ]	**Expresión entre corchetes para una lista de correspondencia que se corresponde con cualquiera de las expresiones representadas en la lista**
{m}	**Se corresponde exactamente m veces**
{m,n}	**Se corresponde al menos m veces pero no más de n veces**
[: :]	**Especifica una clase de carácter y se corresponde con cualquier carácter de esa clase**
\	**Puede tener 4 significados diferentes: 1. Se representa a sí mismo. 2. Presenta elsiguiente carácter. 3. Introduce un operador. 4. No hace nada**
+	**Se corresponde con una o más incidencias**
?	**Se corresponde con cero o una incidencia** **Se corresponde con cualquier carácter del juego de caracteres soportado, excepto NULL**
()	**Expresión de agrupamiento, que se trata como subexpresión única**
[==]	**Especifica clases de equivalencia**
\n	**Referencia a expresión anterior**
[..]	**Especifica un elemento de intercalación como, por ejemplo, un elemento de varios caracteres**

13.2.2. Uso de Metacaracteres

Una correspondencia sencilla.

Problema: buscar 'abc' en una string

Solución: 'abc' encuentra: abc no retorna: 'def'

En el segundo ejemplo, el carácter any se define como un '.'. En este ejemplo se busca el carácter "a" seguido de cualquier carácter, seguido del carácter "c".

Problema: buscar 'a', seguida de cualquier carácter y una 'c' en una string

Solución: 'a,c'

encuentra: abc

encuentra: adc

encuentra: acc

no retorna: 'def'

no retorna: 'abb'

Puede buscar también listas de caracteres sin correspondencia. Una lista de caracteres sin correspondencia le permite definir un juego de caracteres para los que una correspondencia no es válida. Por ejemplo, para buscar cualquier cosa menos los caracteres "a," "b" o "c", puede definir "^" para indicar una no correspondencia.

Expresion: [^abc]

encuentra: abcdef

encuentra: ghi

No retorna: abc

Para hacer corresponder cualquier letra que no esté entre "a" e "i", puede utilizar:

Expresión: [^a-i]

encuentra: hijk

encuentra: lmn

no encuentra: abcdefghi

13.2.3. Funciones de Expresiones regulares

La base de datos Oracle 10g proporciona un juego de funciones SQL que se pueden utilizar para buscar y manipular cadenas mediante expresiones Regulares. Puede utilizar estas

funciones en cualquier tipo de datos que contenga datos de caracteres como, por ejemplo, CHAR, NCHAR, CLOB, NCLOB, NVARCHAR2 y VARCHAR2. Una expresión regular debe ir entre comillas simples. Esto asegura que toda la expresión sea interpretada por la función SQL y puede mejorar la legibilidad del código.

REGEXP_LIKE: Esta función busca un patrón en una columna de caracteres. Utilice esta

función en la cláusula WHERE de una consulta para devolver las filas que se correspondan con la expresión regular que se especifique.

REGEXP_REPLACE: Esta función busca un patrón en una columna de caracteres y sustituye cada incidencia de ese patrón por el patrón que se especifique.

REGEXP_INSTR: Esta función busca en una cadena una incidencia especificada de un patrón de expresión regular. Hay que especificar qué incidencia se desea buscar y la posición inicial desde la que buscar. Esta función devuelve un entero que indica la posición en la cadena en la que ha encontrado la correspondencia.

REGEXP_SUBSTR: Esta función devuelve la subcadena real que se corresponde con el patrón de expresión normal que se especifique.

13.2.4. Sintaxis de la Función REGEXP

```
REGEXP_LIKE ( srcstr, pattern [,match_option] )
REGEXP_INSTR(srcstr, pattern [, position [, occurrence
  [, return_option [ , match_option]]]] )
REGEXP_SUBSTR (srcstr, pattern [, position [, occurrence [,
match_option ]] ] )
REGEXP_REPLACE ( srcstr, pattern [ , replacestr [, position
```

[, occu rrence [, ma t ch_op t i on]]]])

srcstr	**Valor de búsqueda**
pattern	**Expresión normal**
occurrence	**Incidencia que se buscará**
position	**Punto de partida de la búsqueda**
return_option	**Posición inicial o final de la incidencia**
replacestr	**Cadena de caracteres que sustituye al patrón**
match_option	**Opción para cambiar la correspondencia por defecto; puede incluir uno o más de los siguientes valores:** **"c" —utiliza una correspondencia sensible a mayúsculas/minúsculas (por defecto)** **"I" —utiliza una correspondencia no sensible a mayúsculas/minúsculas** **"n" —permite el operador de correspondencia con cualquier carácter** **"m" —trata la cadena de origen como varias líneas**

13.2.5. Realización de Búsquedas Básicas

En esta consulta, en la tabla EMPLOYEES, se muestran todos los empleados cuyos nombres contienen Steven o Stephen. En la expresión utilizada,

' ^S te (v | ph) en$ ' :

- ^ indica el inicio de la sentencia
- $ indica el fin de la sentencia

• | ind i ca o

```
SELECT f i rs t_name , las t_name
FROM emp l oyees
WHERE REGEXP_L I KE ( f i rs t_name , '^S te (v |ph ) en$ ' ) ;
```

13.2.6. Comprobación de la Presencia de un Patrón

En este ejemplo, la función REGEXP_INSTR se utiliza para buscar la calle con el fin de encontrar la ubicación del primer carácter no alfabético, independientemente de si está en mayúsculas o minúsculas. La búsqueda se realiza sólo en las calles que no empiecen por un número. Observe que [:<class>:] implica una clase de carácter y se corresponde con cualquier carácter de esa clase; [:alpha:] se corresponde con cualquier carácter alfabético.

Se muestran los resultados.

```
SELECT s t ree t _add ress ,
REGEXP_ INSTR (s t ree t_add ress , '[^[:a lpha :]] ' )
FROM      l oca t i ons
WHERE REGEXP_ INSTR (s t ree t _add ress , ' [^ [ :a l pha :] ] ' )> 1 ;
```

En la expresión utilizada en la consulta '[^[:alpha:]]':

- [inicia la expresión
- ^ indica NO
- [:alpha:] indica la clase de carácter alfabético
-] finaliza la expresión

Nota: El operador de clase de carácter POSIX le permite buscar una expresión dentro de una lista de caracteres que sea miembro de una clase de carácter POSIX específica. Puede utilizar este operador para buscar un formato específico como, por ejemplo, caracteres en mayúscula, o bien puede buscar caracteres especiales como, por ejemplo, dígitos o caracteres de puntuación.

Se soporta todo el juego de clases de carácter POSIX. Utilice la sintaxis [:class:] donde class es el nombre de la clase de carácter POSIX que se debe buscar. Las siguientes expresiones regulares buscan uno o más caracteres consecutivos en mayúsculas: [[:upper:]]+.

13.2.7. Ejemplo de Extracción de una Subcadena

En este ejemplo, los nombres de calle se extraen de la tabla LOCATIONS. Para ello, se devuelve el contenido de la columna STREET_ADDRESS que está antes del primer espacio mediante la función REGEXP_SUBSTR. En la expresión utilizada en la consulta '[^]+ ':

```
SELECT REGEXP_SUBSTR(street_address , ' [^ ] + ' ) " Road "
FROM loca t i ons ;
```

- [inicia la expresión
- ^ indica NO
- indica espacio
-] finaliza la expresión
- + indica 1 o más
- indica espacio

13.2.8. Sustitución de Patrones

En este ejemplo se examina COUNTRY_NAME. La base de datos Oracle reformatea este patrón con un espacio después de cada carácter no nulo de la cadena. Se muestran los resultados.

```
SELECT REGEXP_REPLACE( countr y_name, '(.)', '\1 ')
"REGEXP_REPLACE"
FROM coun t r ies ;
```

13.2.9. Expresiones Normales y Restricciones de Control

Las expresiones normales también se pueden utilizar en restricciones de control. En este ejemplo, se agrega una restricción de control en la columna EMAIL de la tabla EMPLOYEES.

Esto asegurará que sólo se acepten las cadenas que contienen un símbolo "@". Se prueba la restricción. La restricción de control se viola porque la dirección de correo electrónico no contiene el símbolo necesario. La cláusula NOVALIDATE asegura que no se comprueben los datos existentes.

```
ALTER TABLE emp8
ADD CONSTRA INT ema i l_add r
CHECK ( REGEXP_L IKE (ema il , '@ ' ) ) NOVAL IDATE ;
```

13.3. Práctica

1. Escriba una consulta para buscar en la tabla EMPLOYEES todos los empleados cuyos nombres empiecen por "Ne" o "Na".

2. Cree una consulta que elimine los espacios de la columna STREET_ADDRESS de la tabla LOCATIONS en la visualización.

3. Cree una consulta que muestre "St" sustituido por "Street" en la columna STREET_ADDRESS de la tabla LOCATIONS. Procure que no afecte a ninguna fila que ya contenga "Street". Muestre sólo las filas que se vean afectadas.
4.
5.
6.